

DigiRadio

Open-Source Hi-Fi DAB+/FM Receiver

Technical Manual

Michele Bigi

2026 • Hardware: CERN-OHL-S v2 • Firmware: Apache-2.0

<https://github.com/manvalan/DigiRadio>

Contents

1	Introduction	4
1.1	What DigiRadio is	4
1.2	Who this manual is for	4
1.3	Licensing summary	5
2	Hardware Overview	6
2.1	Design philosophy	6
2.2	System block diagram	6
2.3	The tuner: Si4684	7
2.3.1	Control interface	7
2.3.2	Firmware image upload	7
2.4	The audio DSP: ADAU1701	8
2.4.1	Control and program loading	8
2.4.2	Click-free parameter updates	8
2.5	The Bluetooth module: FSC-BT1035	8
2.5.1	Control interface	8
2.6	The host: ESP32-S3	9
2.7	Bus architecture	9
2.8	Power	9
2.9	RF and antennas	10
2.10	Inter-chip connections (GPIO map)	10
2.10.1	System and control lines	10
2.10.2	Si4684 tuner (SPI)	10
2.10.3	ADAU1701 DSP (I ² C)	10
2.10.4	Device-identity EEPROM (24AA025E48)	11
2.10.5	FSC-BT1035 Bluetooth	11
2.10.6	Audio bus (I ² S)	11
2.10.7	Completeness	12
3	Firmware Architecture	13
3.1	Design principles	13
3.2	Layered structure	13

3.3	Audio signal path	14
3.4	Chip control and boot	15
3.5	Configuration, storage, and user interface	15
3.6	Error-handling model	16
3.7	Toolchain	16
4	Class Reference	17
5	Building and Flashing	18
5.1	Toolchain	18
5.2	Device build	18
5.3	Host unit tests	18
5.4	Documentation	19
6	Licensing	20
6.1	Hardware	20
6.2	Firmware	20
6.3	Documentation and project pages	20
6.4	Third-party components	21

CHAPTER 1

Introduction

DigiRadio is an open-source, high-fidelity digital radio receiver. It receives DAB+ and FM broadcasts, processes the audio through a dedicated signal processor, and streams the result over Bluetooth using a high-resolution codec. The whole project — hardware and firmware — is released as open source for the maker and audio community to study, build, and improve.

1.1 What DigiRadio is

Most DAB+/FM hobby projects stop at basic reception. DigiRadio adds two things that lift it into hi-fi territory: a real audio DSP stage for equalisation and mixing, and a premium-codec Bluetooth output. The result is a genuine wireless audio source rather than a bench demo, and, being fully documented, a practical reference for anyone learning multi-chip audio design or RF-aware PCB layout.

At a glance

A four-chip design — Si4684 tuner, ADAU1701 DSP, ESP32-S3 host, and an FSC-BT1035 Bluetooth transmitter — on a six-layer board, powered from USB-C, configured through an elegant web interface.

1.2 Who this manual is for

This manual documents the system for someone building, flashing, or extending DigiRadio: the hardware at a block level (Chapter 2), the firmware architecture (Chapter ??), the per-class reference that grows with the code (Chapter 4), and how to build and flash (Chapter 5). The exact register-level programming of each chip lives in the source code and its generated API documentation; this manual explains the design and the reasoning behind it.

1.3 Licensing summary

The hardware is licensed under CERN-OHL-S v2 and the firmware under Apache-2.0; full details are in Chapter 6. The project repository is at <https://github.com/manvalan/DigiRadio>.

CHAPTER 2

Hardware Overview

This chapter describes the board in detail: the design choices behind it, each integrated circuit and how it is controlled, the bus architecture, and the inter-chip connections. The complete hardware design — schematics, PCB, Gerbers, and bill of materials — is published in the repository under the CERN-OHL-S v2 licence.

2.1 Design philosophy

Every major choice on the board serves audio quality and reproducibility.

The four-chip split

Rather than a single all-in-one radio IC, DigiRadio separates concerns: a dedicated tuner for reception, a dedicated DSP for audio conditioning, a dedicated Bluetooth module for the wireless codec, and a general-purpose host to coordinate them. Each stage can be understood, measured, and improved on its own.

The consequences of that split run through the rest of the board: a control host that speaks several buses, a clean power tree that keeps the tuner's analogue supplies away from digital noise, and careful RF layout so the two 2.4 GHz radios do not interfere.

2.2 System block diagram

Four devices are coordinated by the ESP32-S3. The tuner produces the audio stream, the DSP conditions and mixes it, and the Bluetooth module transmits it; the host manages all three and provides the network interface.

2.3 The tuner: Si4684

Device	Role	Control bus	Firmware
Si4684	DAB+/FM tuner	SPI	host-loaded image
ADAU1701	Audio DSP (EQ + mixer)	I ² C	host-loaded RAM
ESP32-S3-WROOM-1	Host, Wi-Fi/BLE	—	application FW
FSC-BT1035	Bluetooth 5.2 (aptX Ad.)	UART (AT)	vendor FW + config

Table 2.1: Principal integrated circuits, their control bus, and how each receives its firmware or configuration.

The Si4684 is a single-chip DAB+/FM receiver front-end. It delivers the decoded audio stream to the DSP and reports signal-quality metrics to the host.

2.3.1 Control interface

The device is controlled through a command/response protocol over its host bus. The host issues a command, then polls a clear-to-send (CTS) status bit before reading the response; the driver validates that status byte before trusting any returned payload.

SPI for the tuner

The tuner needs a large firmware image loaded at every power-up (Section 2.3.2). SPI offers substantially higher throughput than I²C, so on DigiRadio the tuner uses **SPI**, keeping boot time short, while the low-rate DSP control uses a separate I²C bus.

2.3.2 Firmware image upload

The Si4684 holds no application firmware of its own at reset. At power-up the host loads a bootloader/patch and then the firmware image for the desired mode (FM or DAB). These images are large, so the driver streams them to the chip in bounded chunks read from the ESP32's flash, rather than buffering a whole image in RAM.

Datasheet

The exact power-up, patch, image-load, and boot command sequence follows the Si468x programming guide (AN649). No command byte is issued without a documented source, and each step in the driver cites its section.

2.4 The audio DSP: ADAU1701

The ADAU1701 is a SigmaDSP audio processor. In DigiRadio it performs equalisation and mixes the tuner audio with the ESP32 audio source before the signal reaches the Bluetooth stage.

2.4.1 Control and program loading

The DSP is controlled over I²C. Its signal-processing program is designed in SigmaStudio and exported as a sequence of register writes (control settings, program RAM, and parameter RAM).

RAM boot, no EEPROM

The board deliberately omits the ADAU1701 self-boot EEPROM. Instead, the ESP32 writes the exported program into the DSP's RAM at *every* boot. This keeps the audio processing under host control and field-updatable — a new EQ or mixer configuration is a firmware update, not a chip re-programming.

2.4.2 Click-free parameter updates

Runtime changes — adjusting an EQ band, changing a mix level — are applied through the ADAU1701 safeload mechanism, which updates parameter cells atomically between audio samples. Writing parameter cells directly while audio plays would produce audible clicks and is not done.

2.5 The Bluetooth module: FSC-BT1035

The FSC-BT1035 (Qualcomm QCC3056) is a Bluetooth 5.2 audio transmitter with aptX, aptX HD, and aptX Adaptive support — the high-resolution codecs that make the wireless output hi-fi rather than merely convenient.

2.5.1 Control interface

The module is controlled with AT commands over UART, with hardware flow control (RTS/CTS). The host sends commands and parses the module's responses explicitly, treating

timeouts as errors. Some settings are held in the module's own non-volatile memory.

Line-In mode

The initialisation sequence must enable Line-In mode (AT+AUXCFG=1) so the module accepts the wired audio coming from the DSP. Omitting it silently breaks the audio path.

2.6 The host: ESP32-S3

The ESP32-S3-WROOM-1 coordinates the whole system. It loads the tuner and DSP firmware at boot, controls all three companion chips over their respective buses, serves the configuration web interface, and provides Wi-Fi connectivity. Its dual-core processor and generous RAM comfortably handle the firmware images and the network stack alongside the control tasks.

2.7 Bus architecture

The host speaks several interfaces, each chosen for its traffic:

Bus	Devices	Purpose
SPI	Si4684	fast firmware-image upload, tuner control
I ² C	ADAU1701, 24AA025E48 EEPROM	DSP control, device identity
UART	FSC-BT1035	AT command / response (with RTS/CTS)
I ² S	Si4684, ESP32, ADAU1701, BT1035	audio (ADAU is master)

Table 2.2: Interface assignment and rationale.

2.8 Power

The board is powered from USB-C. An AP63203 buck converter generates the main rail from the USB input, feeding the digital and analogue sections. The tuner's sensitive supplies are derived and filtered separately to keep RF performance clean.

2.9 RF and antennas

Two 2.4 GHz radios are present — the ESP32-S3 and the Bluetooth module. Their antennas are placed on opposite diagonal corners of the board to maximise separation and minimise mutual desense. Each module keeps its antenna keep-out clear of copper on all layers. The PCB is a six-layer stack-up with controlled impedance on the USB and RF sections.

2.10 Inter-chip connections (GPIO map)

This section is the authoritative wiring reference between the ESP32-S3 and the companion chips. The firmware pin definitions (`board_pins.hpp`) must match these tables exactly.

2.10.1 System and control lines

Signal	ESP32-S3 GPIO	Notes
USB D−	GPIO19	native USB
USB D+	GPIO20	native USB
Reset / boot button	GPIO0	strapping pin

Table 2.3: System and control lines.

2.10.2 Si4684 tuner (SPI)

Signal	Si4684 pin	ESP32-S3 GPIO	Notes
SCLK	pin 30	GPIO13	
MOSI	pin 31	GPIO12	
MISO	pin 32	GPIO9	
SSB (CS)	pin 29	GPIO8	chip select, active-low
RSTB#	pin 4	GPIO38	active-low, ext. pull-down
INTB#	pin 3	GPIO39	active-low, ext. pull-up

Table 2.4: Si4684 control and SPI lines.

2.10.3 ADAU1701 DSP (I²C)

Signal	ADAU1701 pin	ESP32-S3 GPIO
SDA	—	GPIO4
SCL	—	GPIO5
RESET#	pin 5	GPIO47 (active-low, ext. pull-up)

Table 2.5: ADAU1701 control (I²C) and reset. The 7-bit I²C address is 0x34 (ADDR0 and ADDR1 tied to GND).

2.10.4 Device-identity EEPROM (24AA025E48)

A 24AA025E48 EEPROM shares the DSP's I²C bus. Besides a small general-purpose memory, it carries a factory-programmed, globally unique EUI-48 identifier, which the firmware can read to give each unit a stable identity — for example a unique Bluetooth name or a device serial number. Its 7-bit address is 0x52 (A0 to GND, A1 to 3V3).

Device	7-bit address	Set by
ADAU1701	0x34	ADDR0, ADDR1 = GND
24AA025E48 EEPROM	0x52	A0 = GND, A1 = 3V3 (no A2 pin)

Table 2.6: I²C address map (shared bus).

2.10.5 FSC-BT1035 Bluetooth

Signal	BT1035 pin	ESP32-S3 GPIO	Direction
SYS_CTL	pin 34	GPIO15	ESP32 → BT
BT_CTS	pin 15	GPIO21	flow control
BT_RTS	pin 16	GPIO14	flow control
BT_RX	pin 14	GPIO40	ESP32 TX → BT RX
BT_TX	pin 13	GPIO41	BT TX → ESP32 RX
RESET	pin 8	GPIO17	ESP32 → BT

Table 2.7: FSC-BT1035 UART, control, and reset lines.

2.10.6 Audio bus (I²S)

The audio path is a chip-to-chip I²S bus with the ADAU1701 acting as master. Table 2.8 lists the chip-to-chip routing; Table 2.9 lists the ESP32's own I²S source lines that feed the mixer.

Signal	Path	ADAU1701 / BT1035 pins
SDATA_IN0	Si4684 → ADAU	MP0 (pin 11)
SDATA_IN1	ESP32 → ADAU	MP1 (pin 10)
SDATA_OUT0	ADAU → BT1035	MP6 (pin 15) → pin 5
LRCLK	ADAU → BT1035	MP4 (pin 8) + MP10 (pin 16) → pin 7
BCLK	ADAU → BT1035	MP5 (pin 9) + MP11 (pin 19) → pin 4

Table 2.8: I²S audio routing (chip-to-chip). The MP4/MP10 and MP5/MP11 pairs are electrically connected on the board.

Signal	ESP32-S3 GPIO	Direction
BCLK	GPIO6	in from ADAU (master)
LRCLK	GPIO7	in from ADAU (master)
SDATA_IN1	GPIO16	out → ADAU MP1

Table 2.9: ESP32 I²S source lines feeding the mixer. The ESP32 is an I²S slave (clocks in), transmitting audio data. SDATA_IN1 lands on ADAU MP1 (pin 10); GPIO16 is the ESP32 module’s physical pin 9.

2.10.7 Completeness

Map complete

The GPIO and I²C maps are complete and match `board_pins.hpp`. No values remain open.

This section is the authoritative wiring reference for firmware bring-up: the pin definitions in the source code must match it exactly.

CHAPTER 3

Firmware Architecture

The DigiRadio firmware runs on the ESP32-S3 and orchestrates three companion chips into a single high-fidelity audio path: the Si4684 DAB+/FM tuner, the ADAU1701 SigmaDSP, and the FSC-BT1035 Bluetooth transmitter. This section describes how the firmware is organised, how audio flows through the system, and how each chip is controlled. The day-to-day coding conventions (style, documentation, review rules) are intentionally kept out of this manual and live in the repository's `CONTRIBUTING.md` and `AGENTS.md`.

3.1 Design principles

Two ideas shape the whole codebase:

- **Strong typing.** Domain quantities (frequency, gain, station identifier, band) are dedicated types rather than raw integers or floats. Invalid states are made unrepresentable: input from the network, UART, or flash is validated once at the boundary into a domain type, and trusted thereafter.
- **Functional core, imperative shell.** All hardware-free logic — coefficient math, boot-image framing, configuration parsing, station-list operations — lives in a pure *core* that compiles and is unit-tested on the host, with no dependency on ESP-IDF. Every access to the hardware (I²C, SPI, UART, flash) lives in a thin *shell* around that core. This keeps the logic testable without a board attached.

3.2 Layered structure

The firmware is organised in three layers, with dependencies pointing inward only (Figure 3.1). The outer shell may depend on the services and the core; the pure core depends on nothing outside itself.

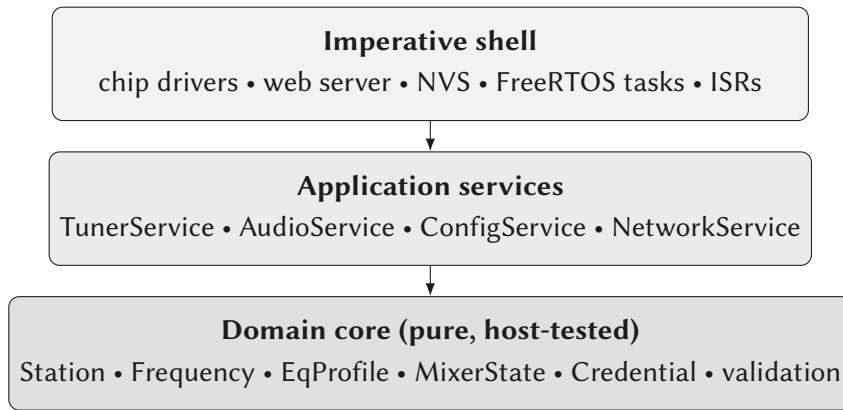


Figure 3.1: Firmware layering. Dependencies point inward; the pure core has no hardware dependencies.

Application services take driver *interfaces* (ITuner, IDsp, IBtModule, ISecureStore) injected at construction. Because the services depend on abstractions rather than concrete hardware, the same logic can be exercised against fakes in host tests and against real silicon on the device.

3.3 Audio signal path

The audio chain is built around sound quality. The Si4684 produces the DAB+/FM audio stream; the ADAU1701 applies equalisation and mixes it with the ESP32 audio source; the FSC-BT1035 streams the result over Bluetooth with aptX Adaptive (Figure 3.2).

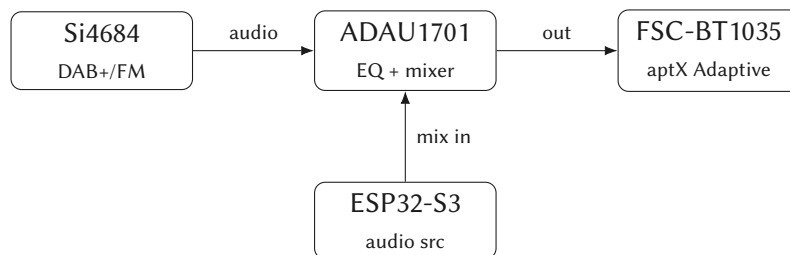


Figure 3.2: Audio signal path. The ADAU1701 input mixer selects/blends the Si4684 and ESP32 sources; equalisation is applied before the Bluetooth output stage.

Equalisation and input mixing are exposed to the rest of the firmware as typed operations — setting an EQ band from a gain, centre frequency, and Q, or setting the mix level of a given source — never as raw DSP cell addresses. The biquad coefficient math runs in the pure core and is verified against reference values in host tests.

3.4 Chip control and boot

Si4684 (tuner). At power-up the driver follows the documented boot sequence: power on, load the patch/bootloader, load the firmware image (FM or DAB), then boot. Firmware images are large and are streamed to the chip in bounded chunks from flash rather than buffered whole in RAM.

Datasheet

The exact power-up, patch, image-load, and boot command sequence follows the Si468x programming guide (AN649). Each step in the driver cites the relevant section; no command byte is issued without a documented source.

The public interface is intent-level (`powerUp`, `loadImage`, `tuneTo`, `readRsq`); register-level detail is private to the driver.

ADAU1701 (DSP). The board carries no self-boot EEPROM. Instead, the ESP32 writes the SigmaStudio-exported program into the DSP's RAM at *every* boot. Runtime changes to equalisation and mixing use the ADAU1701 safeload mechanism, so parameter updates are click-free while audio is playing.

Safeload

Writing DSP parameter cells directly while audio is running produces audible clicks. All runtime EQ and mixer changes must go through the safeload registers.

FSC-BT1035 (Bluetooth). The module is controlled by AT commands over UART. The initialisation sequence includes enabling Line-In mode (`AT+AUXCFG=1`), which is required for the wired audio path from the DSP; command responses are parsed explicitly, with timeouts treated as errors.

3.5 Configuration, storage, and user interface

Network provisioning uses a SoftAP/captive-portal for first-time setup, then joins the configured network as a station. Configuration is exposed through an elegant, minimal web interface served from flash, backed by a typed JSON API; the interface holds no business logic.

Sensitive data — Wi-Fi credentials, user credentials, and the station/frequency list — is

held in encrypted storage at rest. Secrets are wrapped in a dedicated type that cannot be logged or implicitly converted to a string, and buffers are cleared after use.

3.6 Error-handling model

Every operation that can fail returns a typed result (`std::expected<T, Error>`); nothing fails silently and every timeout is an explicit error value. Errors propagate to the layer that can act on them — a driver reports, a service decides (retry, degrade, or surface to the UI), and the top level logs. C++ exceptions are disabled.

3.7 Toolchain

Item	Choice
Framework	ESP-IDF v5.5.x (native)
Language	C++23 (<code>-std=gnu++23</code>)
Error model	<code>std::expected</code> ; exceptions off
Hardware licence	CERN-OHL-S v2
Firmware licence	Apache-2.0

Table 3.1: Firmware toolchain and licensing summary.

The firmware source, build instructions, and development conventions are maintained in the `Software/` directory of the project repository.

CHAPTER 4

Class Reference

This chapter documents the firmware’s public classes at the design level: what each class is responsible for, which collaborators it depends on, its key invariants, and how it fits the system. It complements — and does not duplicate — the generated API documentation, which carries the exact method signatures.

How this chapter grows

Per the development rules, every public, architecturally-significant class (the drivers, the application services, and the public domain-core types) gets a section here, tagged `\label{cls:ClassName}`, added in the same change that introduces the class. A tooling check keeps this chapter in step with the code, so it is always current.

The firmware is under active development. As each public class lands, its section appears below, grouped by layer: domain core, application services, and hardware drivers.

(No public classes documented yet)

Sections will appear here as the firmware is implemented.

CHAPTER 5

Building and Flashing

The firmware is built with ESP-IDF; the hardware-free domain core also builds and is unit-tested on the host machine.

5.1 Toolchain

Item	Choice
Framework	ESP-IDF v5.5.x (native)
Language	C++23 (-std=gnu++23)
Error model	std::expected; exceptions off
Documentation	Doxygen (build must pass)

Table 5.1: Firmware toolchain.

5.2 Device build

Build, flash, monitor

```
idf.py set-target esp32s3
idf.py build
idf.py -p <port> flash monitor
```

5.3 Host unit tests

The pure core is tested on the host, with no board attached. On macOS the tests need a C++23 standard library — use a Homebrew LLVM (≥ 18) or GCC 14 rather than the system Apple Clang.

Host tests

```
cmake -S components/core/test -B build-host \  
      -DCMAKE_CXX_COMPILER="$(brew --prefix llvm)/bin/clang++ \  
      "  
cmake --build build-host  
ctest --test-dir build-host --output-on-failure
```

5.4 Documentation

The API documentation is generated with Doxygen and must build cleanly; an undocumented class, method, or parameter fails the build.

Docs

```
doxygen Doxyfile
```

Keep it green

The Doxygen build and the manual-synchronisation check are part of the definition of done. A change that leaves either failing is not complete.

CHAPTER 6

Licensing

DigiRadio is dual-licensed, with hardware and firmware under separate but compatible open-source licences.

6.1 Hardware

The hardware — schematics, PCB layout, Gerbers, and bill of materials — is licensed under the **CERN Open Hardware Licence Version 2, Strongly Reciprocal (CERN-OHL-S v2)**. Anyone who modifies and distributes the hardware must release their changes under the same terms.

6.2 Firmware

The firmware is licensed under the **Apache License 2.0**. It grants patent rights explicitly and pairs naturally with the ESP-IDF ecosystem, which is Apache-licensed as well. Every source file carries the Apache header with an SPDX identifier.

6.3 Documentation and project pages

Where a hosting platform does not offer the CERN-OHL-S licence in its selector, the closest share-alike option is used for that page, with the authoritative licence stated as CERN-OHL-S v2. The definitive licensing is always the one declared in the project repository.

6.4 Third-party components

The design integrates third-party silicon (Silicon Labs, Analog Devices, Espressif, Feasycom/Qualcomm) whose datasheets, firmware images, and DSP tools remain subject to their respective vendors' terms. Those terms are unaffected by the DigiRadio licences and must be observed when redistributing vendor-supplied binaries.

Manufacturing partner

Prototype fabrication and assembly are supported by PCBWay. Their contribution is acknowledged in the project documentation and on the board.