# SFP Tool Hardware and Software Reference



# SFP Tool Hardware and Software Reference

## Overview

SFP Tool is a field device for reading and programming transceiver modules. It combines

custom hardware, ESP32-S3 firmware, a React and Tauri user interface, and a small firmware recovery web app.

---

# Hardware Architecture

## Main Controller

The hardware is built around an `ESP32-S3`, chosen for:

- Sufficient compute headroom for local transceiver operations.
- BLE connectivity for mobile and wireless desktop control.
- USB support for HID transport and service workflows.
- Available I2C and GPIO resources for display, buttons, power switching, and transceiver presence logic.

## Connectors and User I/O

The hardware design includes:

- One SFP connector.
- One QSFP connector.
- A 1.3 inch `128x64` OLED.
- Front-panel buttons for local navigation.
- USB for host connectivity.
- Battery-backed portable operation on supported revisions.

The firmware supports two OLED controller families:

- `SSD1306`
- `SH1106`

## Hardware Revisions

The firmware currently distinguishes two hardware revisions:

- `1.0`
- `1.1`

Revision `1.1` is the default current revision and includes battery support. Revision `1.0` does not have a battery gauge.

## Relevant MCU Pin Mapping

Current firmware configuration defines:

- `GPIO_I2C_SDA = 21`
- `GPIO_I2C_SCL = 47`
- `GPIO_PWR_SW = 45`
- `GPIO_SFP_PRESENT = 39`

- `GPIO_QSFP_PRESENT = 40`
- `GPIO_QSFP_LPMODE = 35`
- `GPIO_SFP_TX_FAULT = 38`
- `GPIO_SFP_TX = 37`
- `GPIO_BTN_UP = 5`
- `GPIO_BTN_OK = 6`
- `GPIO_BTN_DOWN = 7`

### Battery Management

Battery-aware firmware builds use an `RT9426` battery gauge and enforce low-voltage protection. The configured battery safeguards include:

- Charge detection threshold: `60 mA`
- Critical voltage cutoff: `3000 mV`
- Critical shutdown delay: `3000 ms`

If the battery remains below the configured threshold long enough, the tool powers off to protect the cell.

---

## Hardware Bill of Materials

The PCB export in `pcb/info` contains a machine-generated bill of materials with components such as:

- ESP32-S3 module
- 18650 battery holder
- OLED header
- JST connector
- protection fuse
- buttons
- passives and support circuitry

That file should be treated as the source for manufacturing detail. This reference intentionally stays at the architectural level.

---

## Firmware Architecture

### Platform

The firmware is implemented with Arduino on top of PlatformIO for the ESP32-S3 target.

Primary firmware areas visible in the source tree include:

- transceiver access
- OLED UI
- BLE transport

- USB HID transport
- brute-force password support
- EEPROM storage
- password storage
- script storage and execution
- battery telemetry
- deep sleep and crash recovery
- OTA firmware update support

### Runtime Design Goals

The firmware is designed to execute as much device functionality as possible locally so the tool remains useful without a permanently attached app host.

### Safety and Recovery

The main firmware includes several recovery-oriented behaviors:

- Crash boot counting with a safe-mode threshold.
- Optional disabling of USB or BLE after repeated crashes.
- Stable-boot timers that clear crash counters after successful runtime.
- Minimal safe mode windows for recovery.
- Firmware update state tracking and status reporting.

These mechanisms reduce the chance that a bad build leaves the tool permanently inaccessible.

### Persistent Configuration

The firmware stores tool-specific configuration in ESP preferences, including:

- device name
- OLED brightness
- OLED idle timeout
- OLED display type
- hardware revision

---

## Device Communication Interfaces

### Bluetooth Low Energy

The app communicates with the tool over a custom BLE service:

- Service UUID: `e33cfd9a-883f-4627-b483-76cb710147f0`

The application uses dedicated characteristics for settings, EEPROM access, password workflows, scripting, telemetry, and firmware updates.

### USB HID

Desktop service workflows can also use USB HID transport. This avoids BLE throughput limits and is preferred for large transfers such as firmware upload.

---

# App Architecture

## Technology Stack

The main app is built with:

- React `19`
- TypeScript
- Vite
- Material UI
- Tauri `2`

Desktop builds use Tauri, while Android builds use the generated mobile wrapper. The feature set is intended to stay aligned across supported platforms.

## Main App Pages

The app routes between these major pages:

- Status
- Transceiver
- EEPROM
- Passwords
- Scripts
- About
- Settings
- Special Settings

## Important Behaviors

- The scanner page supports both BLE and USB discovery.
- Windows desktop builds can check for app updates.
- The about page performs firmware upload and exposes build metadata.
- The logs viewer is hidden behind repeated build-number taps.
- The settings menu exposes a long-press path into special settings.

---

# Scripts Subsystem

The scripts feature provides a small domain-specific workflow for deterministic EEPROM programming. Supported command families include:

- field setters such as `name`, `vendor`, `model`, `part`, `revision`, `serial`

- raw byte writes with `bytes`
- password entry with `password`
- checksum recalculation with `checksum`
- final EEPROM commit with `write`

The UI validates syntax before execution and requires `write` to be present.

---

# Password Workflows

The app and firmware support:

- storing passwords on the tool
- changing transceiver passwords
- brute-force password recovery

Password change currently requires both old and new passwords as exactly 4-byte values expressed as 8 hexadecimal characters.

---

# Firmware Update Paths

There are two main update paths:

- In-app OTA upload over BLE or USB.
- Browser-based recovery flashing through the web updater.

The app detects whether an image is app-only or a merged ESP image and skips the bootloader region when needed.

---

# Firmware Recovery Web App

The `software/Firmwareupdater` project is a small web frontend based on `esp-web-tools`. It publishes a browser-based installer that can flash the firmware from a manifest hosted alongside release artifacts.

---

# External Transceiver Library

The repository includes `external/TransceiverTool`, a C++ parser and assembler for transceiver standards data. It contains support code for:

- SFF-8472
- SFF-8636
- related utility and schema files

The app builds a WebAssembly variant so parsing and pretty-printing logic can be reused in the frontend.

## CI and Release Pipeline

The CI pipeline currently covers:

- build-container creation
- MCU firmware build
- updater manifest generation
- desktop app packaging
- Android app packaging
- object publication to S3-compatible storage
- Google Play deployment
- GitLab Pages publication for documentation and updater hosting

Documentation is generated from Markdown during CI and exported as both HTML and PDF.

## Source Tree Guide

Important repository locations:

- `software/MCU/SFP Tool/`: ESP32 firmware
- `software/App/SFP-Tool/`: React and Tauri app
- `software/Firmwareupdater/`: browser recovery updater
- `software/buildenvironment/`: shared CI build image
- `external/TransceiverTool/`: transceiver parsing and formatting library
- `pcb/`: hardware manufacturing and BOM exports
- `docs/`: published documentation sources

## Maintenance Notes

- Update the user manual when workflows or menus change.
- Update this reference when hardware revisions, supported displays, or communication interfaces change.
- Keep CI publishing paths consistent with the manual URLs embedded in the app.