The Pocket FT8 Transceiver Revisited

Jim Conrad, KQ7B, 09/15/25

***Abstract*—*Pocket FT8 Transceiver Revisited* is a derivative of Charles Hill's (W5BAA) Pocket FT8 project [5]. This document describes how to assemble and operate the little rig plus a few hints for modifying it.**

***Index Terms*—Amateur Radio, Digital Signal Processing, DSP, FT8, Homebrew, SDR, Software Defined Radio, Teensy, Si4735, Si5351, Transceiver, WSJTX.**

I. INTRODUCTION & ASSEMBLY

THE *Pocket FT8 Revisited* is a self-contained, single-band, FT8 amateur radio transceiver. Features include:

- 4.0 X 2.8", 4-layer board
- Filter slot for a single band
- FT8 (only) modulation
- 275 mW RF output (QRPP)
- 480x320 TFT touchscreen display
- 600 mHz Teensy 4.1 MCU
- Auto-logging to an SD ADIF file
- JSON configuration file
- "RoboOp" sequenced FT8 QSO
- TCXO clock
- GPS disciplined date, time and grid square
- Operation from a single 5V power brick

To assemble the radio, you need:

- A V2.00 or V3.00 PCB assembly
- A Teensy 4.1
- Adafruit Ultimate GPS PA1616S
- Adafruit 480x320 TFT resistive touchscreen 2050
- Low-profile headers for Teensy
- Tall headers for display
- Right-angle headers for GPS
- SMT BU2032SM holder & CR2032 battery
- SMT SMA antenna jack
- A 5-Pole Low-Pass Filter
- Hand soldered THT red XMIT LED
- T1 bifilar wound xfmr: 10T #26 on FT37-43
- An SD card with CONFIG.JSON file

Note: The GPS header pinout on V2.00 boards is not an exact match for the Adafruit Ultimate GPS; you need to hand-build the GPS cable. V2.00 boards require a patch wire from GPS PPS pin to the GPIO2 (Teensy Header #4) pin.

KQ7B's spectral tests were conducted with the 5-Pole Chebyshev Low Pass filter in .../RFFilters/5Pole40MLPFilter. This 40 meter filter's initial design was created using https://markimicrowave.com/technical-resources/tools/lc-filter-design-tool/ and then hand-tuned with LTspice to place the peak near the 7074 FT8 calling frequency, while optimizing the design for T37-17 inductor cores and standard capacitor values.

The Pocket FT8 Revisited is a homebrew project [6], not a commercial kit. If you're looking for a Heathkit-like experience, consider one of the products from Han's QRP Labs, or the DX FT8. On the other hand, this rig's hardware and the firmware are fully open source and ripe for experimentation. You can use or improve the KiCad schematics and board design, or the PlatformIO firmware and the tests with few restrictions.

II. USER INTERFACE OVERVIEW

The user interface (UI) displays five panels and a row of menu buttons. The upper-left panel provides a waterfall display of received signal strengths, and is also used to change the FT8 offset frequency. The upper-right panel reports the station's status including the current date and time, four character Maidenhead grid square locator, callsign, carrier frequency, and the rig's current activity (e.g. RECEIVE, TUNE, TRANSMIT…). The middle-left panel reports the first 6 messages decoded during the previous FT8 interval. The middle-right panel displays messages directed to or from this station. The bottom panel displays error messages and information about the little rig's activities.

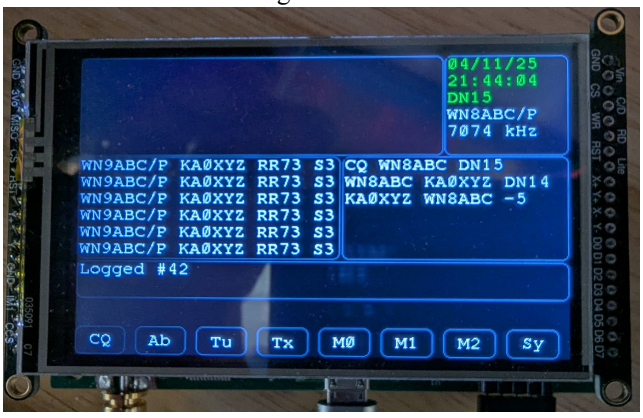


Fig. 1. Pocket FT8 Revisited's User Interface

The menu buttons appear in the bottom row and these control the functions of the FT8 rig.

Jim Conrad, KQ7B, is retired from the computer industry with his wife on their little farm near Grangeville, Idaho (e-mail: conr2286@gmail.com). The many nearby SOTA and POTA sites remain an inspiration for this project.

A. Waterfall Panel

The waterfall provides an interactive display of FT8 activity. Brighter pixels indicate stronger signals while darker pixels reflect weaker signals. The vertical red line indicates your chosen transmitter offset frequency (from the carrier), and may be adjusted by touching the panel. You may wish to choose an offset in a quiet region of the band so your QRPP signals will not be buried beneath those of QRO stations.

B. Station Status Panel

When a GPS fix is available, the GPS-disciplined date and time are displayed in UTC with green text. Without a GPS fix, the date and time values are retrieved from the battery-backed Real Time Clock (RTC) and displayed with yellow text.

Likewise, when a GPS fix is available, the four-character Maidenhead grid square locator is displayed with green text. When a GPS fix is not available, the rig uses the locator provided by the SD configuration file, else the transmitter is disabled if the grid square is not available from any source.

Note that the GPS, while highly desirable, is optional. The rig will "make the best of it" when a GPS fix cannot be obtained. In particular, if a GPS fix was previously obtained after the firmware was loaded into the MCU, the battery-backed RTC will likely be reasonably accurate. Keep in mind that Teensy resets the RTC clock to your computer's local time when you load new firmware into the MCU. To ensure UTC logging and optimal FT8 performance, the rig needs a GPS fix following firmware installation.

The station's callsign appears in the status panel, as does the nominal operating frequency (kHz) and the transmitter's FT8 offset (Hz).

The bottom line of the status panel displays the rig's current mode, RECV, PEND (transmission), XMIT or TUNE.

C. The Decoded Messages Panel

The receiver displays its decoded messages in the middle-left panel. Due to constrained screen space and the conflicting desire to report as much detail as possible about each signal, the display differs from the familiar WSJTX. The Received Signal Level (RSL) appears as S1-S9 rather than the FT8 decibel convention in order to fit the signal report into two characters rather than three. The MCU firmware refreshes the list of received messages following each FT8 time slot. What you see is always the most recent.

Clicking on a received message, e.g. a CQ, instructs RoboOp to engage the remote station in a QSO by transmitting a call in the next available FT8 time slot (this is explained in more detail in the FT8 Essentials section below).

D. The Station Messages Panel

Signals received by and transmitted from the local station appear in the Station Messages panel on the righthand side of the screen. These messages scroll so the oldest message always appears at the top. Most messages appear in white text, but repeated messages appear in yellow rather than in multiple lines. Repeated, unsuccessful attempts to contact a station eventually timeout, and the final attempt appears in grey. Old messages eventually scroll off the panel.

E. The Application Messages Panel

The Application Messages panel displays error and informative messages about the rig's activities. Most are self-explanatory, but a few deserve further attention:

- FATAL: You *must* copy AudioStream6400.h to *.../teensy/hardware/avr/1.59.0/cores/teensy4/AudioStream.h*: This occurs when firmware has been rebuilt with the wrong version of Teensy's AudioStream.h header file. You'll find the correct version in the Extras folder named, AudioStream6400.h, and this must be renamed and copied into the *...cores/teensy4* folder before the rebuilding the firmware.
- ERROR: Unable to access SD card: The SD storage disk containing your `CONFIG.JSON` file (and space for the ADIF log) is not accessible in the Teensy SD slot. Insert your SD card in the Teensy slot, not the Adafruit display board's socket. The rig will continue to boot up but many features are inoperable without your station's callsign.

E. The Menu Buttons

- CQ: Pressing the CQ button instructs the rig to begin calling CQ in the next available FT8 timeslot. If a response is received, RoboOp attempts to engage the responder in a standard, sequenced FT8 QSO, and resumes listening when the QSO completes. If nothing is heard, the rig repeats the CQ message until RoboOp times-out. Pressing the CQ button while the rig is already transmitting CQs terminates CQ activity.
- AB: Abort any transmission in progress.
- TU: Output a steady carrier for antenna tuning. Pressing the TU button during a tuning activity cancels the tuning activity.
- TX: Instructs RoboOp to respond to the first station it hears calling CQ. If successful, RoboOp engages in a standard, sequenced FT8 QSO and then returns to listening. RoboOp repeatedly attempts to contact the heard station until a time-out occurs. You can abort RoboOp's pursuit of a contact by pressing TX again, or by pressing the AB (abort) button. By default, RoboOp will not engage a station that's already in the log; this can be changed in the configuration file.
- M0: Not implemented.
- M1: Not implemented.
- M2: Not implemented.
- Sy: Not implemented.

III. FT8 Essentials

FT8 [1] is a digital communication mode whose excellent weak signal performance imparts some compromises.

All transmissions are of a fixed length and require ~12.6 seconds to complete. Every transmission occurs in one of four time slots that arise at 0, 15, 30 or 45 seconds past the minute. There is, at most, a 2.4 second "dwell" time between the end of a received transmission and the beginning of the following time slot, and that only if the transmission begins exactly on time and can be decoded concurrently as the bits arrive. Timing is important.

Consider the case of a remote station transmitting CQ messages during even-numbered time slots and listening for

responses during odd-numbered time slots. At most, our receiving station has 2.4 seconds to decide to transmit a response during the following odd-numbered time slot while the remote is listening. That's not much time. What's worse is, if our station is too slow to respond in the first available odd-numbered slot, we will have to wait for the beginning of a future odd-numbered time slot to avoid "doubling" with the remote station. If the remote station is rare DX, well… we likely missed our opportunity. Hence… RoboOp.

IV. RoboOp

Because the 2.4 second response time challenges human operators, FT8 software generally implements automation to sequence a standard, contest-like, QSO without human intervention. Pocket FT8 Revisited automates FT8 QSOs with its "RoboOp" feature.

Clicking on a remote station's CQ message directs RoboOp to engage that station. RoboOp replies to their CQ and sequences a QSO with the standardized FT8 messages at the appropriate times. RoboOp retransmits its message when it doesn't hear an appropriate reply.

The TX menu button directs RoboOp to contact the first remote station it hears sending CQ messages. If successful, RoboOp will sequence the entire QSO thereafter.

The CQ menu button directs RoboOp to transmit CQ messages and engage the first responder it hears in a QSO. RoboOp completes, at most, a single QSO; it will never run continuously, logging QSO after QSO, while you enjoy a sandwich and cold beverage from the cooler.

RoboOp employs a configurable time-out determining how long it persists retransmitting a message without receiving a satisfactory response from the remote station.

By default, RoboOp will not respond to CQ messages from a known station already in the log. However, see `CONFIG.JSON` for the override as this is exactly what you may wish to do when testing your latest enhancement to the hardware or firmware.

V. Logging

Pocket FT8 Revisited records an ADIF log file on the Teensy MCU's SD card. By default, the file is called, `LOGFILE.ADIF` but the actual name is configurable. You currently must remove the SD card to copy the log file to your computer.

VI. Configuration

You must supply a configuration file, `CONFIG.JSON`, on the SD card installed in the Teensy MCU. The Examples folder contains a sample CONFIG.JSON which you can modify for your own station. The *callsign* and *frequency* fields are required, the others are optional. If you plan to operate without a permanent GPS connection, you will need to configure the *locator* field with your Maidenhead grid square.

VII. Building the Rig

Pocket FT8 Revisited is a github project at, *https://github.com/conr2286/PocketFT8Xcvr*. The firmware is located in the PocketFT8XcvrFW folder, and the hardware in PocketFT8XcvrHW.

A. Firmware

The firmware design requires a change to the Teensy 4.1 AudioStream.h file, and a copy of the modified file you need resides in the Extras folder named `AudioStream6400.h`. You will have to locate Teensy 4.1's `AudioStream.h` file and replace it with the renamed `AudioStream6400.h` file. No, you can't fudge this. If you try, the firmware will tell on you (you might ask how I know that;).

In pursuit of improved testing, much of the V2.* firmware has been moved from the Arduino 2.0 IDE to PlatformIO, a more capable IDE for projects of this size. The `platformio.ini` file is located in the PocketFT8XcvrFW folder. The easiest way to get started is to install the Arduino 2.0 IDE first (PlatformIO will use the tools and libraries it installs), Arduino's support for the Teensy 4.1 board, the Adafruit GPS Library V1.7.5, the PU2CLR SI4735 V2.1.8 library, the bblanchon ArduinoJson V7.3.0 library, and the Adafruit GFX V1.12.0 library. Some of the libraries build with compiler warnings; the PocketFT8XcvrFW code builds clean as of April 23, 2025.

As of V3.00, the choice of amateur band (e.g. 40M) is hardwired in the PocketFT8XcvrFW.cpp source code. If operation is planned on other than 40 meters, you will need to modify the definitions for MINIMUM_FREQUENCY and MAXIMUM_FREQUENCY for the band of your choice.

To open the project with PlatformIO, first install Visual Studio Code (VSC), then the VSC extension for C/C++ programming, and then the PlatformIO extension. Click on the PlatformIO "alien" icon, then "Pick a folder" then the "PocketFT8XcvrFW" folder. PlatformIO will likely busy itself installing the necessary board, framework and libraries either before or when you click the build (checkmark) icon. The project files should build clean (no warnings); the library files not so much.

It might be possible to build the firmware using the Arduino IDE 2.00. Install the required Teensy 4.1 board and library dependencies (these are listed in PlatformIO.ini). Rename PocketFT8XcvrFW.cpp to become PocketFT8XcvrFW.ino, then move all the src, lib and include folders' files into the PocketFT8XcvrFW folder to satisfy the Arduino's expectations. Don't forget to replace Teensy's AudioStream.h file as described above. Let me know if it works.

Compatibility: The V3.00 firmware runs on either the V2.00 hardware (with the GPS PPS patch wire) or on the V3.00 hardware (without any patch wires).

B. Test Programs

There are a number of folders within the BenchTests folder containing small (mostly Arduino 2.0 IDE) test projects for the hardware. The folders are numbered in a suggested order, and the code they contain was developed for testing new boards. You may find them useful for testing new boards, but they are not required.

Additional tests reside in the PlatformIO test folder. As of the V2.* builds, these mainly address the user interface.

C. Investigations

The Investigations folder contains folders and files used to prove (and disprove!) some of the concepts developed for the firmware and hardware.

D. Filters

The RFFilters folder contains a variety of LTSpice filter designs, simulations and daughter board designs. The transmitter requires a 5th order Chebyshev low-pass filter to meet the USA FCC spectral purity requirements. The receiver might benefit from a bandpass filter, especially when operated near an AM broadcast antenna, but this has not been evaluated as of April, 2025.

E. Hardware

The PocketFT8XcvrHW folder contains the KiCAD 9 files for the schematic and PCB artwork. The Bill of Materials (BOM) is in the Mfg folder. PCBWay built and assembled most SMT components on my boards. The only SMT components not in the BOM (like… the antenna jack, the RTC's battery holder…) are really easy to solder. I installed low-profile headers on the Teensy 4.1 and THT soldered these to the PCB.

F. GPS

J3 on the V2.00 boards break-out a 5-pin connection for the GPS. The PCB layout for J3 was designed for an obsolete GPS and the pinout doesn't exactly align with the Adafruit Ultimate GPS PA1616S; that means you will need to construct the GPS cable by hand rather than use a flat ribbon. The firmware needs the Pulse Per Second (PPS) signal to determine when the GPS has acquired a fix. The V2.00 boards require a patch wire connecting PPS to Teensy GPIO2.

The firmware will run without a GPS, *but* a working GPS supplies the maidenhead grid square, UTC date, and UTC time for the ADIF log, as well as synchronizing the transceiver with the FT8 time slot boundaries. If you don't have a GPS, the Teensy loader sets the MCU's date/time to that of your development computer. This is sometimes accurate enough to conduct an FT8 QSO but is far from optimal for either FT8 or logging. I strongly recommend using a GPS in the field.

G. Auxiliary Analog/Digital Expansion

J4 on the V3.00 boards break-out additional Teensy analog and digital connections for a planned expansion of the firmware.

H. Enclosure

The FreeCAD and STL files reside in the `Enclosure` folder. There is one file to build the main enclosure, and a second for the display bezel. As of July, 2025, the enclosure requires substantial post-processing to make it usable.

VII. Limitations

Several menu buttons are defined but unimplemented in Version 2.* firmware.

The FT8 library does not currently support hashed callsigns. Received traffic with hashed callsigns is not displayed. RoboOp will not respond to hashed callsigns.

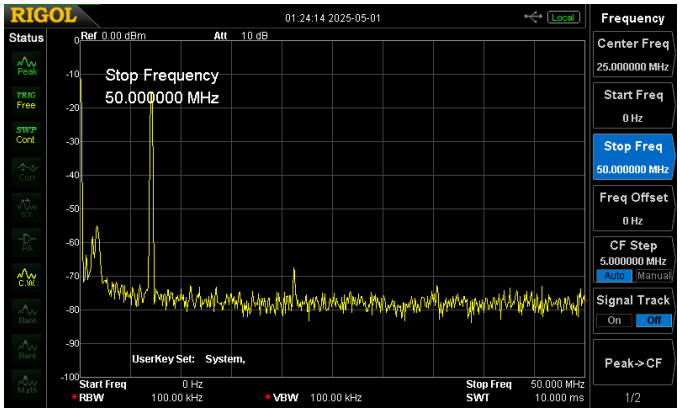
I have long expected disappointment with the Si4735 receiver chip but its real-world performance seems up to the task of communicating with "ordinary" stations receiving the little rig's 275 mW QRPP signal. This level of performance was only achieved through careful board revision to eliminate Si4735 interference from digital noise. Long-term, the receiver may migrate to a Tayloe detector to facilitate QRPP-to-QRPP QSOs.

VIII. Bug Reports and Other Contributions

This is a homebrew, personal project, without commercial support. Perhaps the best way to report a bug is as a github issue. As with any project, please review the known issues (https://github.com/conr2286/PocketFT8Xcvr) prior to submitting a duplicate report.

If you wish to submit a github pull request to incorporate your bug fix or enhancement upstream, please contact me at conr2286@gmail.com to discuss your proposal.

IX. Spectral Purity



The analysis was obtained during modulation with the carrier at 7074 kHz using the 40M 5-Pole Chebyshev low-pass filter. Note the strongest displayed spur lies *below* the fundamental, in the neighborhood of about 200 kHz, and is at least ~40 dB down. The 2nd harmonic is in the noise (consistent with the PA's square wave output) and the 3rd ~50 dB down. The harmonics are no problem and the 200 kHz spur easily cleansed by a resonant antenna, or trivial high-pass filter that also reduces AM broadcast noise into the receiver. This spur deserves further investigation.

Acknowledgments

Pocket FT8 Revisited [6] is a derivative of Charles Hill's (W5BAA) Pocket FT8 project [5] with important contributions from Ricardo Caritti (the Si4735 library), Karlis Goba (YL3JG, the FT8 library), Barb (WB2CBA, the SN74ACT244 PA), and many other widely available Adafruit, Arduino and PJRC libraries. Charles and Barb continue their work with the DX FT8 project [7], another Pocket FT8 derivative. I am indebted to tips from now-forgotten bloggers (Hans and others) for optimizing the phase noise performance from the Si5351, optimizing the sensitivity of the Si4735

receiver, combined RF/AF/digital PCB design guidelines, and getting the most from the Teensy 4.1 MCU. With regard to the Si4735, I cannot overemphasize the importance of good PCB design and isolating the chip on its own I2C bus as the Si4735 seems quite noise-sensitive.

Back in the 60s, the ARRL published an amazing receiver for its day, the Junior Miser's Dream, that accomplished so much with so little. Its focused design influenced my engineering career, discouraging unwarranted complexity. The Pocket FT8 revisits that theme in a digital world. May you too stand on the shoulders of giants.

REFERENCES

[1] S. Franke, B. Somerville and J. Taylor. "The FT4 and FT8 Communication Protocols," *QEX*, July-August 2020.

[2] SKYWORKS. "AN619 Manually Generating an Si5351 Register Map for 10-MSOP and 20-QFN Devices," September 2021.

[3] SKYWORKS. "AN1234 Manually Generating an Si5351 Register Map for 16QFN Devices," December 2021.

[4] VE2ZAZ. "VE2ZAZ Si5351 Synthesizer Board," https://ve2zaz.net/Si5351_Synth/Si5351_Synth.htm, November 2019.

[5] Charles Hill. https://github.com/Rotron/Pocket-FT8

[6] J. Conrad KQ7B. https://github.com/conr2286/PocketFT8Xcvr

[7] Charles Hill. https://github.com/WB2CBA/DX-FT8-FT8-MULTIBAND-TABLET-TRANSCEIVER