

T41 Digital Modes (FT8/WSJT-X) Integration

T41 Digital Modes (FT8 / WSJT-X) — Teensy USB Audio + CAT to Raspberry Pi 5

This procedure adds digital-mode capability (FT8 to start with, WSJT-X's other modes later) on top of the existing Pi 5 + Touch Display 2 display/telemetry build in `INTEGRATION_PROCEDURE.md`. It does **not** replace or change anything in that document — the PiLink UART link (3 wires: GND, Pi TXD -> Teensy RX1, Pi RXD -> Teensy TX1) keeps doing exactly what it does today.

The good news, found while researching this: the Teensy firmware already has a complete digital-mode audio and CAT transport built in (`USBAudio.cpp` / `.h`, the `DIGITAL_RECEIVE` / `DIGITAL_TRANSMIT` states in `ModeSm.cpp`, and the `DG` / `SR` Kenwood-style CAT commands in `CAT.cpp`). None of that needed to be written for this — it rides on the Teensy's own USB port, which most builds haven't been using for this yet. This procedure is about *connecting* that existing capability to the Pi, not building it.

1. What's being added

One more physical connection, no new parts:

Connection	Carries	Status
PiLink UART (GND, Pi pin 8 -> Teensy pin 0, Pi pin 10 -> Teensy pin 1)	Frequency/band/mode/S-meter/spectrum telemetry, virtual button presses	Existing, unchanged
Teensy USB -> Pi USB-A (new)	44.1 kHz digital-mode RX/TX audio + Kenwood-style CAT (frequency, PTT, mode), all on one USB cable	This procedure

The Teensy's own USB connector — the same one used to flash the firmware and already wired into the T41 chassis — becomes, when the firmware is built with the right USB Type, a composite device: a USB sound card (for FT8/WSJT-X audio) plus a USB serial CAT port (for frequency/PTT), simultaneously. The Pi sees it exactly the way a Windows or Mac PC would if you plugged the Teensy into it directly — this is the same interface thousands of existing T41/Mustang builds already use to run WSJT-X from a PC; the only difference here is that the “PC” is the Pi 5 sitting inside the radio instead of a separate laptop.

2. Firmware: rebuild with USB Audio enabled

The Teensy sketch already has everything wired (`USBAudio.cpp` / `.h`, the `DIGITAL_RECEIVE` / `DIGITAL_TRANSMIT` states, the `DG1`; / `DG0`; CAT commands), gated behind one compile-time symbol: `AUDIO_INTERFACE`. That symbol is defined automatically by one specific Arduino IDE board setting — nothing in the source needs to change.

1. Open `Mustang2Sketch.ino` in the Arduino IDE (Teensyduino installed).
 2. **Tools -> USB Type -> “Serial + MIDI + Audio”**. This is the setting that defines `AUDIO_INTERFACE`. (Teensyduino has no “Dual Serial + Audio” option — see the comment at the top of `SDT.h` — so this build has only *one* USB serial port, not two.)
 3. **One real tradeoff to know about before flashing:** with `AUDIO_INTERFACE` defined, `CATSerial` becomes the primary `Serial` (shared with CAT), and `Debug()` is compiled out to a no-op so its output can't corrupt the CAT stream (see `SDT.h`). If you rely on `Serial.println()` debug output during normal (non-digital-mode) development, you'll lose that separate debug port in this build. Keep a non-audio build around for firmware debugging, and flash the audio build only when you're working on/using digital modes.
 4. Verify + upload as normal.
 5. After flashing, confirm in the radio's own menu/CAT that `DG1`; / `DG0`; works: entering digital mode should force 176.4 kbps (required — that's the sample rate where the receive chain's Fs/4 tap lands exactly on 44,100 Hz, so the USB audio endpoint needs no resampling) and the radio should return to its previous rate on `DG0`;.
-

3. Physical connection

Nothing new to source — this is a standard USB cable, not a Mouser BOM item:

1. Identify the Teensy 4.1's USB connector (the same one used for programming). Confirm which cable type your T41 chassis brings that connector out to (it may already be extended to a panel-mount USB jack; check your chassis build before assuming it's the bare Teensy connector).
2. Run a cable from that connector to any free **USB-A port on the Pi 5**. Standard USB 2.0 cable, no special shielding or length requirements beyond normal USB practice (keep it away from the BFO/VFO lines and RF deck if your chassis routing brings it close to them — same general RF-hygiene rule as any other digital cable in the chassis).
3. This is **in addition to** the existing PiLink UART wiring and the existing power wiring — none of those change.

See the updated `wiring_diagram.svg / .pdf`: the new connection is drawn in green, separate from the blue PiLink UART line.

4. Pi-side: verify the USB audio + CAT device

Before writing any decode software, confirm the Pi actually sees the Teensy as a sound card and a serial port once it's in digital mode:

1. On the Teensy, enter DIGITAL mode (front panel menu, or send `DG1` over the existing CAT link if you're testing from a terminal first).
2. On the Pi:

```
arecord -l          # should list a USB Audio device (Teensy)
aplay -l           # same device, playback side
ls /dev/ttyACM*     # should show a serial device for the Teensy's CAT port
```

3. Quick audio sanity check — record a few seconds while the Teensy is receiving a signal, then play it back:

```
arecord -D plughw:CARD=Teensy,DEV=0 -f S16_LE -r 44100 -c 1 -d 5 test.wav
aplay test.wav
```

(Card name may differ — check the `arecord -l` output.)

4. Quick CAT sanity check (adjust the device path to what `ls /dev/ttyACM*` showed):

```
stty -F /dev/ttyACM0 38400 raw -echo
echo -n 'FA;' > /dev/ttyACM0 # ask for VFO A frequency
head -c 20 /dev/ttyACM0      # should read back FAxxxxxxxxxx;
```

If this doesn't respond, double check `raspi-config` hasn't also grabbed this port as a login-shell serial console (same class of conflict as the PiLink UART port in `INTEGRATION_PROCEDURE.md` Section 4 — the Pi's own `/dev/serial0` policy doesn't apply here since this is a USB CDC port, not the GPIO UART, but the same principle applies: nothing else should be reading this port).

If both checks pass, the hard part (getting Linux to see genuine radio audio and CAT) is done — everything from here is Pi-side application software.

5. Phase 1 (recommended starting point): FT8-only, built into the existing UI

No VNC, no second desktop app — decode FT8 directly and show it in the same Tkinter window (`display.py`) that already shows VFO/S-meter/spectrum.

1. **Get the real FT8 codec source.** The sBitx firmware download in `Mustang Radio/sbitx-main/` references `ft8_lib` and `clu` as git submodules, but that folder was downloaded as a plain ZIP so the actual codec source isn't present (only stale build artifacts). Get the real source directly:

```
git clone https://github.com/eccloud/ft8_lib.git
```

```
git clone https://github.com/ec1oud/clu.git # optional - callsign/DXCC lookups only
```

`ft8_lib` itself is portable C needing only FFTW3 and pthreads — no GTK, no sBitx-specific hardware code.

2. **Build a small decode helper** (Python or a thin C program) that:

- Opens the Teensy's ALSA capture device (`arecord` -equivalent via `python-sounddevice` or `PyAlsaAudio`) at 44.1 kHz, mono.
- Feeds ~15-second blocks of samples into `ft8_lib`'s decode routines (the same call pattern sBitx's `modem_ft8.c` uses: buffer samples, call the FFT/Costas-sync/LDPC decode pipeline once per FT8 time slot).
- Publishes decoded messages (callsign, grid, SNR, dB, timestamp) — e.g. over a local socket or a small shared file/queue that `radio_ui.py` polls.

3. **Add a new pane in `display.py`** — same pattern as the existing pane-mirroring table in `PiDisplay/README.md` — showing a scrolling table of decodes (time, dB, freq offset, message).

4. **PTT for FT8 transmit:** send `TX;` over the CAT serial port to key up (same command Hamlib already uses for T41 CAT — see `CAT.cpp`'s `TX_write`), play synthesized FT8 tone-audio out the Teensy's USB audio playback device, then `RX;` to unkey. Enter/exit digital mode itself with `DG1;` / `DG0;` around the whole FT8 session (or leave it in digital mode for the full session — either works, matching how you'd operate `DG` manually today).

5. **Frequency awareness:** you already have the dial frequency via the existing PiLink STATUS downlink — no need to query it over CAT too, though either source will agree since both ultimately read the same Teensy state.

This phase proves the audio + CAT path end-to-end and gets you decoding FT8 inside the radio's own UI, without touching WSJT-X at all.

6. Phase 2 (later): full WSJT-X, in its own remote-viewable window

Once Phase 1 has proven the audio/CAT path, this phase swaps in the real WSJT-X application (every mode: FT8, FT4, JT65, MSK144, etc.) instead of a custom decoder, using the same remoting approach the sBitx firmware already uses for exactly this purpose (`sbitx-main/web/scripts/start_wsjtx.sh`, `start_novnc_proxy.sh`) — that scripting is generic Linux desktop-remoting (Xvfb + x11vnc + noVNC), not sBitx-hardware-specific, so it ports directly.

1. Install WSJT-X, Hamlib, Xvfb, x11vnc, and a noVNC/websocketify package on the Pi.
2. Configure WSJT-X's **Radio** settings: Rig = a Hamlib "Kenwood"-family rig (the T41's `CAT.cpp` implements standard Kenwood `FA / FB / MD / TX / RX` commands at 38400 baud — the same command set Hamlib's Kenwood backends expect). Point it at the Teensy's `/dev/ttyACM*` CAT port.
3. Configure WSJT-X's **Audio** settings: pick the Teensy's USB audio device for both Soundcard In and Out.
4. **One manual step WSJT-X doesn't know about:** send `DG1;` over the CAT port before starting a WSJT-X session (and `DG0;` after), since entering T41's digital mode is a T41-specific command, not something WSJT-X sends on its own. A small wrapper script around WSJT-X's launch/quit can handle this automatically.
5. Adapt `start_wsjtx.sh`: run it on a virtual display (`Xvfb :1`), capture with `x11vnc`, serve via `novnc_proxy` — copy the script's structure from `sbitx-main/web/scripts/`, updating only the paths (it's currently written for sBitx's own install layout under `/home/pi/sbitx/`).
6. The result: a real WSJT-X window, reachable and clickable from any browser on your network, running independently of the Tkinter `radio_ui.py` window.

7. Summary: what's confirmed vs. what's new work

Piece	Status
Teensy digital-mode USB audio transport (<code>USBAudio.cpp/.h</code>)	Already built — found in the current firmware
Teensy digital-mode state machine (<code>DIGITAL_RECEIVE / TRANSMIT</code>)	Already built
Teensy CAT commands (<code>DG</code> , <code>SR</code> , <code>TX / RX</code> , standard Kenwood <code>FA</code> etc.)	Already built

USB Type build setting to enable it (<code>Serial + MIDI + Audio</code>)	Confirm/set this — Section 2
Physical USB cable, Teensy -> Pi	New wiring — Section 3, no new parts
Pi-side ALSA/CAT verification	New — Section 4
FT8 decode integrated into <code>display.py</code>	New software — Section 5
WSJT-X + noVNC remote window	New setup, but the scripts already exist in <code>sbitx-main/web/scripts/</code> — Section 6