

T41 Teensy 4.1 ↔ Raspberry Pi 5 — UART Protocol Design

Updated 2026-10-04 for Mustang 2 V1.0.34. The link has grown from the original three downlink messages to fourteen, plus virtual-encoder, button-repeat and menu-jump uplinks. V1.0.34 is built and posted on the website; V1.0.26 onward are newer than the long-running V1.0.25 bench build. Per-version history is in `MUSTANG_2/CHANGELOG.md`; build steps are in `INTEGRATION_PROCEDURE.md` (Section 9 covers everything added since the first power-up).

Context

Hybrid architecture: the Teensy 4.1 keeps all real-time audio/DSP/RF control exactly as-is. A Raspberry Pi 5 (2GB) is added to drive the display/UI on an official Raspberry Pi Touch Display 2 (7”), plus a LAN-only browser remote, with a future goal of running WSJT-X/FT8. The link between the two boards is a simple UART called **PiLink**.

Board history: Pi 5 (full replacement) → hybrid Teensy + Pi 5 → Pi Zero 2 W (unavailable) → Pi Zero W → Pi 3 Model B → **Pi 5 (final)**, for the CPU/RAM/USB headroom the digital-modes goal needs, plus native DSI support for the Touch Display 2.

Display history: buydisplay.com ER-TFTM070-5V4 (RA8875 + FT5316, SPI/I2C) → **Raspberry Pi Touch Display 2, 7” (final)** — a standard DSI panel Raspberry Pi OS drives as a normal framebuffer + 5-point capacitive touch device.

Design principle

The Pi is an enhancement, not a dependency. If it’s unplugged or crashes, the radio keeps working off its physical front panel (buttons/encoders/key/PTT), which stays wired directly to the Teensy.

Clicks and touches on the Pi (or in the browser remote) are fed into the **same** input paths the physical panel uses: - buttons → `SetButton()` with a real `BUTTON_*` id (`VIRTUAL_BUTTON`) - knobs → the same `iFILTER_*`, volume, `iCENTERTUNE_*` and `iFINETUNE_*` interrupt events the encoder ISRs post (`VIRTUAL_ENCODER`)

The Pi-side app does its own hit-testing and only sends these semantic events across.

Link settings

- Teensy `Serial1` (pins 0/1), **921,600 baud**, 8N1.
- `InitializePiLink()` adds a **1024-byte transmit buffer** (`addMemoryForWrite()`) so a whole spectrum frame fits; without it every spectrum frame was silently dropped (V1.0.4).
- Pi 5 device:** `/dev/ttyAMA0`, **not** `/dev/serial0`. On the Pi 5, `/dev/serial0` points at the debug-connector UART, opens without error and receives nothing.

Framing

```
[0x7E][0xA5][MSG_TYPE][LEN_LO][LEN_HI][...payload...][CRC-8]
```

CRC-8 (poly 0x07) over `MSG_TYPE` + `LEN_LO` + `LEN_HI` + payload, computed incrementally on both sides so no payload-size buffer is needed. On a CRC failure the receiver drops the frame and scans forward for the next `0x7E 0xA5`. The Teensy parser is non-blocking, one byte at a time, so it never threatens the main loop’s timing.

Downlink: Teensy → Pi

ID	Message	Rate	Payload / purpose
0x01	STATUS	10 Hz	<code>PiLinkStatusMsg</code> : frequency, band, mode state, TX/RX (from <code>modeSM.state_id</code> , not the raw PTT pin — V1.0.7), power level, filter bandwidth, keyer WPM, SAM offset, passband low/high edges, visible spectrum span (<code>lowerFreqHz/upperFreqHz</code>), band legal limits (<code>bandLowHz/bandHighHz</code>), Menu position (<code>menuPrimaryIndex/menuSecondaryIndex</code> , V1.0.15)
0x02	SMETER	20 Hz	<code>int16</code> , 0.1 dB units (<code>GetSMeterDBM()</code>)
0x03	SPECTRUM	~15 fps	512 bytes, one per bin, computed with the Teensy’s own <code>pixelnew()</code> plus the noise-floor shift, so Menu → Display spectrum scale/floor affect the Pi exactly as they

			affect the Teensy (V1.0.11)
0x04	AUDIO_SPECTRUM	~15 fps	Live audio-FFT bars (mirrors <code>ShowSpectrum()</code> 's <code>audioYPixel[]</code>)
0x05	SETTINGS	2 Hz	<code>PiLinkSettingsMsg</code> : everything on the Settings pane (Vol/AGC/Mic/STn slot, AGC, tune/fine-tune increments, RX/TX attenuation, auto-notch, noise reduction, DSP gain, antenna, zoom, sample rate, CPU temp and load, firmware version string), <code>decoderOn</code> , <code>audioVolume</code> , <code>secondScreenActive</code> , and (V1.0.17-V1.0.22) <code>spectrumScale</code> , <code>spectrumFloor</code> , <code>spectrumFloorAuto</code> , <code>spectrumFillTraceEnabled</code> . 58 bytes
0x06	PARAMETER	10 Hz, only while adjusting	<code>PiLinkParameterMsg</code> : the red “Label: value” box, sent while <code>uiSM.state_id == UISm_StateId_UPDATE</code>
0x07	RXIQ_CAL	10 Hz, while that screen is open	<code>PiLinkRXIQCalMsg</code> : mirrors Menu > Calibration > Receive IQ (V1.0.23)
0x08	BIT_RESULTS	2 Hz, while that screen is open	<code>PiLinkBITMsg</code> : mirrors Menu > Diagnostics > “I2C BIT test” results, one pass/fail per board (V1.0.25)
0x09	EQUALIZER	10 Hz	<code>PiLinkEqualizerMsg</code> : Menu > Audio Options > Adjust Equalizers - RX/TX side, selected cell, increment, 14 RX + 14 TX levels. 32 bytes (V1.0.26)
0x0A	TXIQ_CAL	10 Hz on screen, 2 Hz off	<code>PiLinkTXIQCalMsg</code> : Menu > Calibration > Transmit IQ - band, flags (dual VFO, PTT, Auto running, direct-coupled), increment, attenuation, frequency, per-band amp/phase/IRR/dBc/DC offsets. 168 bytes (V1.0.30)
0x0B	FREQ_CAL	10 Hz on screen, 2 Hz off	<code>PiLinkFreqCalMsg</code> : Menu > Calibration > Frequency - modulation, band, correction factor, increment, SAM error, frequency. 18 bytes (V1.0.30)
0x0C	POWER_CAL	10 Hz on screen, 2 Hz off	<code>PiLinkPowerCalMsg</code> : Menu > Calibration > Power - band, PA, power-cal state, transmitting, units, attenuation, measured/target power, increment, recorded points, SSB point, per-band <code>Psat/k/gain</code> (float32). 212 bytes (V1.0.30)
0x0D	WATERFALL	~15 fps, half an interval after SPECTRUM	512 bytes: the T41's own waterfall colour index per column (<code>230 - pixelold[x]</code> , 0..116 into <code>gradient[]</code>), so the Pi's waterfall colours match the T41 exactly (V1.0.32)
0x0E	CW_DECODE	10 Hz decoding, 2 Hz otherwise	<code>PiLinkCWDecodeMsg</code> : active, locked, WPM, running decoded-character count, the T41's 15-character decode window, plus the CW settings page values (SNR, AT Grace, Mode, Key Type, Dit, AvgD/D, Upd Age, Coeff). The Pi uses the count to build a longer line under the waterfall, and shows the CW settings page while Decode is on. 42 bytes (V1.0.33)

Bandwidth: spectrum (512 B) and audio spectrum at ~15 fps plus the smaller messages are still well within the ~92 KB/s a 921,600-baud link carries.

Uplink: Pi → Teensy

ID	Message	Payload	Purpose
0x10	VIRTUAL_BUTTON	1 byte: a <code>BUTTON_*</code> id	Fed into <code>SetButton()</code> , the same path as a front-panel press
0x11	HEARTBEAT	none	Lets the Teensy notice the Pi went away (2 s timeout); the Pi sends one every 0.5 s
0x12	VIRTUAL_ENCODER	2 bytes: encoder id, direction (0 = decrease, 1 = increase)	One click of a knob. Encoder ids: 0 Filter, 1 Volume, 2 Tune (main VFO), 3 Fine-tune
0x13	VIRTUAL_BUTTON_REPEAT	2 bytes: button id, count	Presses the same button N times in one go, so jumping several steps in a

			dropdown (Zoom, Band, Mode...) is near-instant (V1.0.16)
0x14	MENU_SELECT	2 bytes: primary, secondary index	Jumps straight to a Menu item (JumpToMenuItem()), only from Home and not while transmitting (V1.0.28)

Calibration screen buttons. The calibration screens use the normal button ids and encoders, exactly as in each screen’s switch in `Loop.cpp`. Some ids have an extra job on those screens only: - Receive IQ: 13 `DECODER_TOGGLE` = Stop Auto, 14 `DfE` = Exit without saving. - Transmit IQ (V1.0.30): 13 = Stop Auto (releases PTT), 1 `MAIN_MENU_UP` = Exit without saving. - Frequency and Power (V1.0.30): 1 = Exit without saving.

Calibration > S meter level is an ordinary Menu value and uses `PARAMETER`.

How the Pi uses it

- Toolbar dropdowns** (Menu, Vol, Band, Mode, Demod, Course Inc, Fine Inc, Zoom, BW, Noise, Decode, Home) are built from the firmware’s own tables and send the button presses and encoder clicks the physical panel would.
- Menu selection** computes Filter-encoder clicks from the Menu’s real position in `STATUS` (V1.0.15). Before that the Pi kept its own guess, which drifted whenever the front panel moved the Menu and made remote selections land back on Home.
- Mouse wheel** sends `VIRTUAL_ENCODER` clicks: Filter while a `PARAMETER` box is showing, otherwise whichever knob was last claimed from Vol/BW, or Tune (Shift = Fine-tune) in the browser remote.

Browser remote and audio (Pi side, not on the UART)

- `web_relay.py` (port 8765, passkey-gated, **LAN-only, plain ws://**) re-broadcasts the same parsed telemetry to browsers and forwards their commands through the same code the touchscreen uses.
- `audio_relay.py` captures RX audio from the **Teensy’s own USB port** (USB Type “Serial + MIDI + Audio”, 44.1 kHz) and streams it to the browsers. This path is separate from PiLink; it’s the same USB audio/CAT path planned for FT8/WSJT-X.

Power (Pi + display branch)

Tap the 12 V distribution rail with its own fused branch; common ground only, never tied to the Teensy board’s own 5 V regulator. Recommended: Geekworm RPi5-5V5A-PD (terminal-block variant, real USB-C PD, 5 V/5 A, 5 A ATO fuse). Mean Well SD-25A-5 / SD-15A-05 GPIO-powered alternates are in the Integration Procedure. The Pi and display share one feed via the display’s power cable on Pi pins 2/6.

UART pin assignment

`PI_LINK_SERIAL = Serial1` (Teensy pins 0/1) is valid because this board uses the **Voltage reversal prevention** power option; the alternate **Shutdown routine activation** option (ATtiny85, U4) uses the same two pins, and every other Teensy 4.1 hardware UART is already taken on this board. On this build the PiLink wires plug into U4’s empty socket, which is wired to pins 0/1.

Pi side: pin 8 (GPIO14, TXD) → Teensy pin 0 (RX1); pin 10 (GPIO15, RXD) → Teensy pin 1 (TX1); plus a dedicated ground wire. Serial console disabled in `raspi-config` (hardware enabled, login shell off).

Implementation status (V1.0.34)

- Teensy:** `PiLink.h/PiLink.cpp` in the Mustang 2 sketch, started from `InitializePiLink()` and polled every main-loop pass. Running.
- Pi:** `pilink_client.py` (protocol), `display.py` (Tkinter UI), `radio_ui.py` (main app), `cal_screens.py` (Transmit IQ / Frequency / Power calibration screens, V1.0.30), `web_relay.py` + `web/remote.html` / `web/remote_mobile.html` (browser remotes), `audio_relay.py` (RX audio), `config.py`, `desktop/` launchers.

Open items

- ~~Source vs. flashed firmware~~ **Resolved 2026-10-01 (V1.0.25):** every struct in `PiLink.h` now matches `pilink_client.py` field for field (`STATUS` 44, `SETTINGS` 58, `PARAMETER` 37, `RXIQ_CAL` 3, `BIT_RESULTS` 7 bytes), including `audioVolume/secondScreenActive`. The V1.0.30 calibration structs (`TXIQ_CAL` 168, `FREQ_CAL` 18, `POWER_CAL` 212 bytes) were checked against `pilink_client.py` with a host compile of the structs.
- FT8 / WSJT-X over the Teensy USB audio + CAT path (see `FT8_WSJTX_INTEGRATION.md`).
- Browser remote: SWR and State of Health panes; PTT time-out and a control lock before allowing transmit from the browser.
- A protocol version handshake so a Pi/Teensy mismatch is reported clearly.
- Ethernet (PJRC kit + QNEthernet) remains the upgrade path if the UART ever runs out of room.