

ATU-100 I2C Display Tap — Feasibility & Wiring

Oct 18, 2026

What this is

A feasibility review and wiring plan for mirroring the ATU-100 Extended Board's OLED readout (PWR, SWR, L, C) onto the Pi 5's own screen and both web remotes, via a passive tap on the ATU-100's existing I2C display bus — no changes to the ATU-100's PIC firmware, and the ATU-100 keeps working exactly as it does today whether or not the Pi is listening.

Verdict: feasible, and better than a plain bitmap mirror. The ATU-100's open-source firmware confirms the display runs on slow, bit-banged software I2C (not the PIC's hardware peripheral) at roughly 20 kbit/s — far below standard 100 kHz I2C — and draws characters from a fixed, fully-known 5×8 font table at fixed screen positions. That combination means a Pi-side listener can do exact digit-for-digit text decoding of PWR/SWR/L/C (not just copy pixels), and has generous timing margin to keep up doing it in software.

Why it's feasible

- I2C/SSD1306 OLED, confirmed from the official manual and the open firmware source (github.com/Dfinitski/N7DDC-ATU-100-mini-and-extendedboards).
- Bus runs on the PIC's RB6/RB7 pins (the same pair used for ICSP programming), bit-banged in software with roughly 5µs per bit-phase — about 20 kbit/s effective, not the 100/400 kHz a hardware I2C peripheral would use. Slow and forgiving for a software-timed listener.
- The firmware's `oled_wr_str()` renders ASCII text to pixel columns via a fixed 5×8 font table (`font5x8.h`) before transmitting — only pixel bytes ever cross the wire, but the table is exact and public, so a sniffed pixel stream decodes back to the exact character with a simple table lookup, no OCR guesswork involved.
- The live status screen's layout is fixed, not scrolling or menu-driven: PWR=, SWR=, L=...uH and C=...pF are always written at the same page/column on the 128x64 OLED (`main.c`, `led_type 4/5`), confirmed by the schematic photo's own readout format. That lets the decoder key off fixed coordinates instead of parsing a moving layout.
- The Pi 5's I2C1 header pins (GPIO2/GPIO3, physical pins 3/5) are genuinely free — confirmed against the existing PiLink UART map, which uses GPIO14/15 instead.

Power — the ATU-100 wants 12V direct, not 5V

The official manual specs the Extended board's supply as **10–15V DC, max 400mA (typically 150–200mA)** — that's the board's whole input, relays included. The schematic's 78L05 regulator only steps that down locally, on-board, to the 5V the PIC and OLED logic need; it isn't the board's external supply voltage.

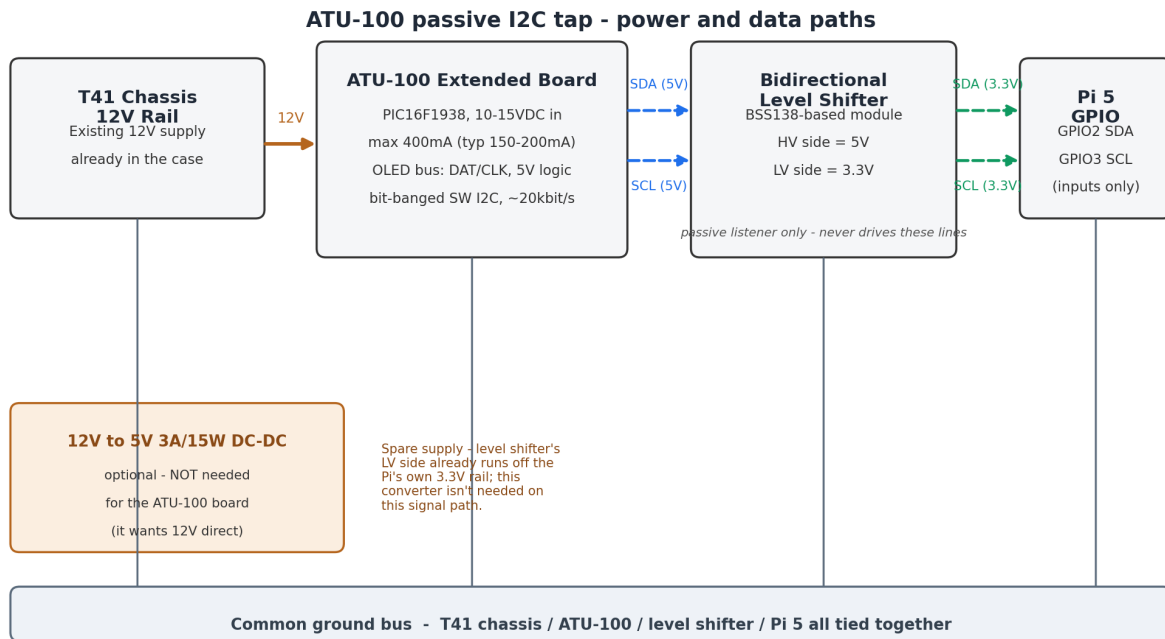
About the 12V→5V 3A/15W converter

That converter would under-power the ATU-100 if wired to its main input — the board needs 12V there directly, not 5V. Tap the T41's existing 12V rail straight to the ATU-100's own DC input instead; no converter needed on that path. The level shifter's low-voltage (3.3V) side already runs off the Pi's own 3.3V rail, so the converter isn't needed there either. It's a fine spare to have on hand, just not part of this particular tap.

Parts list

Part	Spec	Qty	Notes
Bidirectional logic level shifter	BSS138-based module, 4+ channel (e.g. Adafruit/SparkFun style)	1	HV side to ATU-100's 5V bus, LV side to Pi's 3.3V GPIO
Hookup wire	22–26 AWG stranded, a few colors	~2 ft	SDA, SCL, 5V ref, 3.3V ref, ground
12V tap / fuse-tapped lead	Matches T41's existing 12V rail wiring	1	Feeds the ATU-100's DC input directly
Connector for ATU-100 DAT/CLK pins	Matches the board's ICSP/PROG header pitch	1	Tap only — never insert a device that could drive these lines
Multimeter	—	—	Confirm 5V bus voltage and continuity before powering up
Logic analyzer or scope (recommended, not required)	Any cheap USB logic analyzer (e.g. 8-channel, 24MHz+)	1	Makes bringing up the sniffer software far faster than debugging blind

Wiring diagram



The Pi's GPIO side is wired as inputs only and never drives SDA/SCL — the PIC is the bus's only master and has no awareness of a second device, so the tap has to stay purely passive. All four grounds (T41 chassis, ATU-100 board, level shifter, Pi 5) are bonded to one common reference.

Dashed arrows: logic-level sides of the tap, passive listen-only. Solid orange arrow: 12V power to the ATU-100 board itself, unrelated to the signal tap. Vertical grey lines: shared ground bus.

Software plan

- **Sniffer.** A small program (Python via a GPIO edge-callback library, or C if Python framing proves unreliable) watches SDA/SCL as inputs, reconstructs START/STOP and byte framing from the edges, and reassembles the OLED's page-addressing and data writes — same approach as any I2C bus analyzer, just built for this one bus.
- **Decoder.** Because the font table and the fixed field layout (PWR=/SWR=/L=.../C=...) are both known exactly from the firmware source, reconstructed pixel columns at known page/column positions are reverse-matched against `font5x8.h` to recover the literal PWR, SWR, L and C values as text/numbers — not a picture of the screen.
- **Integration.** Once decoded, this follows the same pattern already used for the Digital/FT8 and Calibration screens: a new toolbar button opens an overlay on the Pi's own Tkinter display, and the same values are broadcast over the existing `web_relay.py` WebSocket relay so the desktop and mobile web remotes show the identical readout with no separate implementation.

Not done yet — recommended build order

- 1 Wire the physical tap: 12V from the T41 rail straight to the ATU-100's DC input; level shifter HV side to the ATU-100's DAT/CLK pins; level shifter LV side to Pi GPIO2/GPIO3; common ground bonded across all four. Confirm with a multimeter before powering anything up.
- 2 Bench-verify the tap is truly passive: power the ATU-100 up alone first and confirm its own OLED still updates normally with the Pi disconnected, then reconnect and confirm no change in ATU-100 behavior.
- 3 Write and bench-test the GPIO sniffer against the live bus, ideally cross-checked against a logic analyzer capture of the same transactions.
- 4 Build the font-table decoder and verify it reproduces PWR/SWR/L/C correctly against the ATU-100's own OLED, side by side.
- 5 Wire the result into a Pi toolbar button, overlay screen, and `web_relay.py` broadcast, following the existing Digital/Calibration screen pattern.

No wiring, level shifter purchase, or sniffer/decoder code exists yet — this document is the feasibility finding and build plan to work from before any of that starts.