Clockwork PicoCalc — Deep Dive

Assembly, hardware, firmware, programming stacks, and gotchas

Jeff Swan · Hack Tools project

2026-05-02

Clockwork PicoCalc — Deep Dive

> Authoritative reference for assembly, hardware, firmware, programming stacks, and gotchas. Companion docs: ../CLAUDE.md (project state), ../DEVELOPMENT.md (toolchain), ../01-about-me/bringup_checklist.md (day-one playbook). Last revised 2026-05-02.

This doc is variant-agnostic — calls out Pico 1 (RP2040) vs Pico 2 (RP2350) vs Pico 2 W (RP2350 + Wi-Fi/BLE) wherever they diverge. Items I couldn't confirm during research are marked **(unconfirmed)** so they don't propagate as fact.

Table of contents

1. TL;DR
2. Kit contents
3. Assembly walkthrough
4. Hardware tour
5. Default firmware & the multi-boot SD system
6. Programming stacks
7. Display & keyboard programming
8. PIO state machines on PicoCalc
9. Audio
10. Project ideas (broken out by focus area)
11. Known issues & errata
12. Resources

---

1. TL;DR

The PicoCalc is a $75 retro-style handheld kit. A swappable Raspberry Pi Pico module sits in a socket on the ClockworkPi v2.0 carrier alongside a 4″ 320×320 IPS LCD, a 67-key backlit QWERTY keyboard, dual speakers, an SD slot, an AXP2101 PMU, and 8 MB external PSRAM. The keyboard is driven by an STM32F103 coprocessor that the Pico talks to over I^2C. The kit ships flashed with a multi-boot SD bootloader and ~8 ready-to-run UF2 images

(PicoMite BASIC, uLisp, FUZIX, NES emulator, MP3 player, Phyllosoma, picolua, Picoware) — pick one at boot and it loads from the card without USB.

Programming target = whatever the swappable Pico in the socket is. Default kit ships with **Pico 1 H (RP2040)**, but the same case takes **Pico 2 (RP2350)** or **Pico 2 W (RP2350 + CYW43439 Wi-Fi/BLE)**. Latest **Pico SDK 2.x** supports all three via PICO_BOARD=pico|pico2|pico2_w.

---

2. Kit contents

What ships in the standard 2026 box (per official kit page; confirm against the parts list inside):

- ClockworkPi **v2.0** mainboard (carrier PCB)
- **Raspberry Pi Pico module** pre-fitted in the socket. Default is Pico 1 H (RP2040, 264 KB SRAM, 2 MB flash). Some current SKUs ship Pico 2 (RP2350) or Pico 2 W instead — read the silkscreen on the module.
- **4-inch 320×320 IPS** display, SPI interface, with tempered-glass cover
- **QWERTY backlit keyboard** (membrane + STM32F103-driven controller PCB) with number row, function keys (F1–F10), four-way D-pad, ESC/BREAK
- **Dual speakers**
- **ABS plastic shell halves** + 2.5 mm hex key
- **32 GB microSD** pre-loaded with the official multi-boot bundle
- **Not included:** 2× 18650 lithium-ion cells, USB-C cable

---

3. Assembly walkthrough

Vendor claims a few minutes with no soldering. Real assembly is mostly screw-driving — but there are three failure modes that have killed units on the forum, so don't skim:

3.1 Battery selection

- 2× 18650 cells, 18 mm Ø ±0.5, 65–69 mm length.
- **Brand-name only.** Samsung 35E (3500 mAh) or 30Q (3000 mAh, higher current draw), Panasonic NCR18650B/G, LG MJ1. Anything claiming ≥4000 mAh is fake.
- Buy from **18650batterystore.com**, **illumn.com**, or **IMR Batteries**. Do not trust Amazon or AliExpress for these.
- The on-board AXP2101 provides over-voltage / over-current / over-temp protection, so unprotected cells are acceptable. Protected cells fit if total length ≤69 mm.
- Flat-top OR button-top both fit; flat-top requires extra polarity care (no ridge to bias against).

3.2 Battery polarity (IMPORTANT)

Multiple forum reports of users reverse-installing one cell and bricking the AXP2101. Check the silkscreen markings *carefully* — they're not always strongly contrasted. The AXP2101 has limited reverse protection.

If the PMU is killed, replacement is possible but requires fresh eFuse configuration — see forum thread "AXP2101 PMU eFuse configuration after replacement" before attempting.

3.3 Mainboard + keyboard + LCD

- **LCD ribbon**: lift the FFC connector latch, insert with contacts down, latch closed. Don't force.
- **Speaker JST**: keyed but verify before mating.
- **Keyboard FFC**: another lift-latch-and-seat. Make sure it's all the way home.
- **Pico module**: pre-fitted in the kit. If you ever swap it, the **socket pinout is mirrored from the standard Pico when viewed from the front** — DigitalDreams' corrected diagram lives in forum thread #17456 ("Core socket pinout"). Easy to flip 180° if you don't check.
- BOOTSEL on the Pico is reachable through the **rear grill of the case** (toothpick or thin card).

3.4 Smoke test

With cells seated and shells closed, long-press the power button (top-right of keyboard) ~1 second. Backlight comes on, splash shows PicoMite (default UF2) or the multi-boot menu. If nothing: USB-C may charge the cells enough to boot in a few minutes.

---

4. Hardware tour

Subsystem	Detail
MCU	Whichever Pico is in the socket: Pico 1 / Pico 2 / Pico 2 W. See §4.1.
Keyboard MCU	STM32F103R8T6, scans the matrix, manages the power button, LCD backlight, keyboard backlight, and battery telemetry. **I^2C slave at 0x1F.** Bus speed must be 10 kHz for reliable comms (per DeepWiki).
Keyboard FW	Source under Code/picocalc_keyboard/ in the official repo. Build with Arduino IDE + STM32duino, or STM32CubeIDE. GPL v3. Latest stable BIOS = **1.4**; uLisp v1.1 currently only runs on **1.2** — don't auto-upgrade if Lisp matters.
PMU	X-Powers **AXP2101** — single-cell PMIC with on-board E-Gauge fuel gauge, 4 DC-DC converters, 11 LDOs, OVP/OCP/OTP. Reverse-polarity is **not** fully protected.
PSRAM	**8 MB** on the carrier, memory-mapped via the Pico's

	QSPI/XIP. RP2350 has a dedicated XIP+PSRAM bus making this faster on Pico 2. Pico 2 W has on-module PSRAM too — collision/coexistence is **(unconfirmed)**; check the forum before betting on both.
Display	4.0″ 320×320 IPS, **SPI**. Driver IC commonly cited in community drivers as **ILI9488** (320×320 region driven via partial-window mode); some older docs say "ILI9486." Confirm by reading the SPI init sequence in picocalc-text-starter/drivers/lcd.c. Frame rate ~30–60 Hz depending on SPI clock.
Frame buffer	320×320 × 16 bpp = **204,800 bytes**. Does **not** fit alongside meaningful program state in RP2040's 264 KB SRAM — community C drivers render in stripes/tiles or use the 8 MB PSRAM. Fits trivially in RP2350's 520 KB.
Audio	Stereo PWM only — left=GP26, right=GP27. They share a PWM slice → both channels are forced to the same frequency, so true stereo is impossible without an external I^2S DAC. (Forum #18339 "Audio with PIO PWM" has a workaround.)
SD slot	**SPI-attached** (not SDIO). Default layout = MBR with FAT32 main partition + 32 MB raw tail partition (FUZIX root). Cards >32 GB work if reformatted FAT32 (factory exFAT must be replaced).
USB	USB-C on the case routes to the Pico module's native USB. Appears as the Pico's BOOTSEL drive + serial port. Charging via VBUS goes through the AXP2101.
GPIO break-out	Side-header exposes the Pico's free GPIO not consumed by display/SD/keyboard/audio. Per-pin map: see DigitalDreams' diagram in forum #17456. 3.3 V rail measures ~0.9% low under load on his unit — use external Vref for ADC accuracy.

4.1 Pico variants — what changes

	Pico 1 (RP2040)	Pico 2 (RP2350)	Pico 2 W (RP2350 + CYW43439)
Cores	2× Cortex-M0+	2× Cortex-M33 + 2× Hazard3 RISC-V (pick at boot)	same as Pico 2
SRAM	264 KB	520 KB	520 KB
Flash	2 MB	4 MB	4 MB
Wireless	none	none	Wi-Fi 4 (2.4 GHz) + BLE 5.2

Frame buffer fits in SRAM?	No (use stripes or PSRAM)	Yes (with room to spare)	Yes
PICO_BOARD=	pico	pico2	pico2_w
Pico SDK version min	any 1.x or 2.x	**2.x**	**2.x**
TrustZone-M	—	yes	yes
Notable extras	—	XIP cache, PSRAM bus, FPU	+ CYW43 driver in SDK

Practical implications: **for any C app that wants a full-screen frame buffer, Pico 2 is much easier**. Pico 1 forces you into stripe-rendering or PSRAM tricks. For wireless work (Bruce/Marauder-style projects, Picoware, Wi-Fi tools, BLE, MQTT) you need Pico 2 W.

4.2 Upgrading the Pico module

The ClockworkPi v2.0 mainboard's Pico **socket** is the design's secret weapon — the Pico is not soldered, so you can swap modules in <60 seconds without touching firmware on the carrier. This makes the upgrade path low-risk and cheap. Useful to understand before you commit to one.

4.2.1 Why upgrade — concrete reasons

Stay on Pico 1 H (the default) if:

- Your projects are text/menu/REPL-driven (PicoMite, uLisp, FUZIX, picolua, terminal apps).
- You're not pushing graphics beyond stripe rendering.
- You don't need wireless.
- You want maximum community-tested code paths — Pico 1 has the deepest install base.

Upgrade to Pico 2 (RP2350, no wireless) when you hit one of:

- *Frame buffer pressure.* You want to write a graphics-heavy app — emulator, paint program, scope UI, sprite-heavy game — and stripe rendering is making the code ugly. Pico 2's 520 KB SRAM lets `uint16_t fb[320][320]` (~200 KB) live as a normal global with 300+ KB still free for state.
- *FPU / DSP work.* RP2040 has no hardware FPU; every float multiply is a software call. Pico 2's M33 has single-precision FPU + DSP extensions — anything signal-processing-flavored (FFT for audio, demodulation for an SDR-companion role, real-time filtering) gets 5–20× faster.
- *More PIO.* RP2040 has 8 state machines across 2 PIOs; RP2350 has 12 across 3. Matters when you're combining a logic analyzer + PIO PWM + custom protocol on the same device.
- *Larger code/asset budget.* 4 MB flash vs 2 MB. Doubles room for fonts, sprite atlases, ROMs in flash, embedded data.

- *Faster XIP / dedicated PSRAM bus.* Random reads from the carrier's 8 MB PSRAM are noticeably faster than going through RP2040's QSPI XIP. If you're using PSRAM for full FB, this is real.
- *RISC-V option.* Pico 2 lets you boot Hazard3 RISC-V cores instead of M33. Mostly curiosity territory unless you have a specific use.
- *Future-proofing.* TrustZone-M, secure boot, glitch detector. Lab toys today, useful tomorrow.

Upgrade to Pico 2 W (RP2350 + CYW43439) when:

- Everything Pico 2 gives, *plus* you want Wi-Fi 4 (2.4 GHz, 802.11n, single-band) or BLE 5.2.
- You want to run **Picoware** (the bundled wireless-tools custom firmware) without external modules.
- You want to build any of the §10.3 wireless project ideas — Wi-Fi auditing on your own networks, MQTT telemetry, BLE sniffing, an OTA update path, network-time-synced apps.
- You want to drop the deep-dive's "Wi-Fi/BLE on side-header module" workaround and just use what's on board.

Don't upgrade if you only need wireless occasionally — an external **ESP32 module** or **nRF52** breakout on the side header costs $5–10, gives you better RF specs than the CYW43439, and keeps your Pico module on RP2040 for compatibility with the broadest community library set. Pico 2 W's wireless is convenient, not lab-grade.

4.2.2 What you give up by upgrading

- *Some PicoCalc-specific UF2s lag behind* on Pico 2 / Pico 2 W — the SD bundle's /pico1-apps/ directory is years older than /pico2-apps/. Verify the UF2s you actually use (PicoMite, uLisp, your own apps) all have current Pico 2 builds before you commit. Most do; some hobbyist forks don't.
- *Pico SDK 2.x required.* If you have any locally-built C code that pinned to SDK 1.5 or earlier, it'll need a small port to 2.x. Public starters (picocalc-text-starter, etc.) are already on 2.x.
- *Pico 2 W only:* on-module PSRAM coexistence with the carrier's 8 MB PSRAM is **(unconfirmed)** — read the forum before relying on the full 8 MB.
- *uLisp v1.1 currently only validates on Pico 1.* (Combined with the BIOS 1.2 requirement, Lisp users should keep a Pico 1 H around even if they own a Pico 2 for everything else. The socket makes "swap to Pico 1, boot uLisp, swap back" a 30-second operation.)
- *Power draw.* Pico 2 W with active radio is ~70–250 mA depending on activity vs Pico 1's idle ~25 mA. Battery life on 2× 18650 stays multi-hour, but it's a real difference in always-on scenarios.

4.2.3 Cost & where to buy

Module	Typical price	Source
Pico 1 H	$5	Adafruit (#5525), DigiKey,

		Mouser, PiHut, Pimoroni
Pico 2	$5	Adafruit (#6202), DigiKey, Mouser
Pico 2 H *(headers)*	$7	DigiKey, Mouser
Pico 2 W	$7	Adafruit (#6210), DigiKey, Mouser
Pico 2 WH *(wireless + headers)*	$9	DigiKey, Mouser

Get the H/WH variant — Pico 2 / 2 W ship without headers by default and you'd need to solder them yourself. The PicoCalc socket needs through-hole male headers. Adafruit part numbers above are H/WH where it matters.

4.2.4 Physical swap procedure

The whole operation is sub-minute. ESD precautions (touch grounded metal, work on a dissipative mat if you have one) — Pico modules are robust but not invulnerable.

1. **Power off.** Long-press the keyboard's power button until the screen blanks. Don't trust "asleep" — fully off.
2. **Disconnect USB-C** if plugged.
3. **Open the rear shell** — 2.5 mm hex screws (the same ones from assembly).
4. **Lift the existing Pico module straight up** out of the socket. It's a 2× 20-pin header arrangement; pull evenly on both ends to avoid bending pins. A small flat-head pry can help if it's snug — go gentle, alternate sides.
5. **Check orientation of the new module.** The PicoCalc socket pinout is **mirrored from the standard Pico orientation** when viewed from the front. DigitalDreams' diagram in forum thread #17456 ("Core socket pinout") is the canonical reference. The micro-USB / USB-C end of the Pico module faces the **rear grill** (BOOTSEL access). Get this wrong and you'll short rails — confirm before pressing down.
6. **Seat the new module** evenly. Press down on both header rows simultaneously, not one end at a time. The pins should fully bottom out — about 5 mm of pin showing above the socket is normal.
7. **Don't reattach the shell yet.** First do a smoke test:
   - With cells still in (or USB-C connected for power), long-press power.
   - Backlight should come on.
   - The multi-boot menu should appear (or boot to PicoMite).
   - If the screen stays dark, check orientation — pull the module, rotate 180°, re-seat.
8. **Reattach the shell.**

4.2.5 Software changes after upgrade

Once the module is swapped, three things change:

1. **PICO_BOARD= in your CMake.** Replace -DPICO_BOARD=pico with -DPICO_BOARD=pico2 (Pico 2) or -DPICO_BOARD=pico2_w (Pico 2 W). Rebuild every C/C++ project.

   ```
   cd ~/pico/picocalc-text-starter
   rm -rf build && mkdir build && cd build
   cmake -G Ninja -DPICO_BOARD=pico2 ..    # or pico2_w
   ninja
   ```

2. **SD folder for UF2s.** Pico 2 binaries go in **/pico2-apps/** on the SD card, not /pico1-apps/. The multi-boot loader picks the right folder based on which silicon is detected at boot. UF2s built for pico will not run on pico2 — the Cortex-M0+ instruction set is a strict subset of M33 in many ways but the binary format and boot sequence differ.

3. **MicroPython UF2.** Replace micropython_pico.uf2 with micropython_pico2.uf2 or micropython_pico2w.uf2 from LofiFren's repo. The Python code on top usually works unchanged.

You can keep both pico1-apps and pico2-apps populated on the SD — useful if you swap modules occasionally. The bootloader auto-routes.

4.2.6 Recommended upgrade decision tree

Will any of your real upcoming projects do one of:
(a) full-screen frame buffer with sprites/animation,
(b) FPU-heavy DSP / FFT,
(c) >2 MB embedded assets, or
(d) Wi-Fi/BLE without an external module?
├── No → stay on Pico 1 H. Largest community library set, lowest config friction.
├── Yes (a/b/c only, no wireless) → Pico 2 H. ~$7. Drop-in.
└── Yes (d, plus optionally any of a/b/c) → Pico 2 WH. ~$9. Drop-in.

If you're unsure: **stay on Pico 1 H**. The socket means you can upgrade later in literally one minute when a specific project demands it. Don't pre-emptively buy a Pico 2 W "just in case" — buy it when a concrete project is gated on it.

---

5. Default firmware & the multi-boot SD system

5.1 What ships flashed

- Pico's internal flash: **bootloader_pico.uf2** (uf2loader by pelrun, originally adwuard/Picocalc_SD_Boot — pelrun's fork is now canonical).
- That bootloader looks at the SD card on power-on. If a multi-boot menu config is present and the user holds **Up** (or **F1** / **F5**) at boot, it shows the menu. Otherwise it auto-loads the last-launched UF2.
- Fallback if SD missing or unreadable: PicoMite, also baked in.

5.2 SD layout (since v0.6)

```
/                                FAT32
├── BOOT2040.uf2                 The menu UI itself
├── pico1-apps/                  UF2s for RP2040 boards
│   ├── PicoMite_v6.02.01b4_beta.uf2
│   ├── PicoCalc_uLisp_v1.1.uf2
│   ├── PicoCalc_NES_v1.0.uf2
│   ├── PicoCalc_MP3Player_v0.5.uf2
│   ├── picolua.uf2
│   ├── phyllosoma_kb.uf2
│   ├── Picoware_v1.6.9.uf2
│   └── ...
├── pico2-apps/                  (or merged folder — varies by bundle)
└── (unallocated 32 MB raw)      FUZIX rootfs (last partition)
```

The first entry in [brackets] in the menu = the last-launched "default app." Subsequent power-ons skip the menu unless you hold the Up key.

5.3 Where to download

- Releases: **https://github.com/clockworkpi/PicoCalc/releases** — full SD bundles
- Bootloader source: **https://github.com/pelrun/uf2loader**

5.4 Adding your own UF2 to the menu

Drop the file into /pico1-apps/ (or /pico2-apps/ per variant), eject, reboot. The bootloader scans the directory at boot — no manifest editing required for current revisions.

5.5 Recovering the bootloader after a BOOTSEL flash

If you BOOTSEL-flash a custom UF2 directly to the Pico's flash, you've overwritten bootloader_pico.uf2. To restore the multi-boot menu:

1. Hold BOOTSEL, replug USB, drag bootloader_pico.uf2 from the SD release back onto the BOOTSEL drive.
2. Power-cycle. Multi-boot is back.

This is why **dropping UF2s on the SD card is the preferred iteration loop** once dev is set up — no flash overwrite, no BOOTSEL gymnastics.

---

6. Programming stacks

The PicoCalc is famously promiscuous about languages. What follows is what's actually working today, with install steps. None of these require giving up the others — the multi-boot SD lets you switch by holding Up at power-on.

6.1 Pico SDK in C/C++ — first-class

The canonical low-friction starter is **BlairLeduc/picocalc-text-starter** — its drivers/ folder has battle-tested LCD, keyboard, audio, SD, and STM32 southbridge code with printf/scanf retargeted to display + keyboard. Use this rather than rolling your own.

Install:

```
# Linux deps
sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential ninja-build

# SDK + starter
mkdir -p ~/pico && cd ~/pico
git clone -b master --recurse-submodules https://github.com/raspberrypi/pico-sdk.git
git clone https://github.com/BlairLeduc/picocalc-text-starter.git
git clone https://github.com/clockworkpi/PicoCalc.git    # for keyboard FW source

export PICO_SDK_PATH=$HOME/pico/pico-sdk    # add to ~/.bashrc

# Build, choosing variant:
cd picocalc-text-starter
mkdir build && cd build
cmake -G Ninja -DPICO_BOARD=pico ..        # or pico2, pico2_w
ninja
```

Resulting .uf2 → SD /pico1-apps/ (or /pico2-apps/).

Key wiki: https://github.com/clockworkpi/PicoCalc/wiki/Setting-Up-the-Pico-SDK-on-Linux-for-Pico-Development. See §7 below for the actual driver API.

6.2 MicroPython — best for prototyping

The PicoCalc-aware fork by **Lofi Fren** wraps the display, keyboard, and SD drivers as native MicroPython modules. Three prebuilt UF2s — pick the one that matches your Pico:

- micropython_pico.uf2 (RP2040)
- micropython_pico2.uf2 (RP2350)
- micropython_pico2w.uf2 (RP2350 + Wi-Fi/BLE)

Install: hold BOOTSEL, plug USB, drag the UF2. Connect with **Thonny** (set interpreter = "MicroPython (Raspberry Pi Pico)") or mpremote:

```
pip install --user mpremote
mpremote connect auto repl            # interactive
mpremote cp app.py :main.py           # push to device, runs on boot
```

Repo: **https://github.com/LofiFren/PicoCalc**. Trade-off vs C: ~10–50× slower, no PIO assembler, can't use most pico-sdk helpers directly. Fine for UI/logic; bad for tight DSP or signal work.

6.3 PicoMite (MMBasic) — default firmware

Geoff Graham's PicoMite has an explicit **PicoCalc** target. It's a complete BASIC interpreter with file I/O, graphics primitives, FFT, sound, sprites — the kind of "BASIC + everything you wished BASIC had" that makes it a serious development environment, not just a toy. v6.02.01b4 ships pre-loaded on the SD card.

- Authoritative: **https://geoffg.net/picomite.html**
- PicoCalc-tweaked sources: https://github.com/cuu/PicoMite, https://github.com/madcock/PicoMiteAllVersions

Reach for PicoMite when you want a self-contained on-device dev environment with no host computer in the loop.

6.4 uLisp — Common Lisp on the metal

Active on PicoCalc with display + keyboard support. Currently **only works on BIOS 1.2** of the keyboard FW; BIOS 1.4 compatibility is WIP. If Lisp is a serious goal, hold off on BIOS upgrades.

- UF2: PicoCalc_uLisp_v1.1.uf2 from the releases page
- Build from source: Code/uLisp/ in the official repo
- Docs: **http://www.ulisp.com/show?56ZO=** and forum thread #17107

6.5 picolua — Lua REPL with graphics + SD

Multiple maintained forks. Most active right now is Lana-chan/picocalc_lua (RP2350 support added; commit 4778a8f notes a low-power-sleep hang as a known issue).

- https://github.com/cuu/picocalc_lua (original)
- https://github.com/Lana-chan/picocalc_lua (RP2350)
- https://github.com/benob/picocalc_lua (alt)

6.6 Rust — embassy-rp + rp-hal

Both crate ecosystems target both RP2040 and RP235x. There is no PicoCalc-specific BSP crate that I could confirm — the awesome-pico-calc list has a "Rust driver" entry but the link wasn't surfaceable during research, so treat it as **(unconfirmed)**. Practically: start from a Pico/Pico 2 BSP, port the LCD + keyboard drivers (the C code in picocalc-text-starter is small enough to translate).

- https://github.com/embassy-rs/embassy
- https://github.com/rp-rs/rp-hal

6.7 TinyGo — Go on the Pico

Confirmed working — the forum has a worked example: "Hello World and RPN Calculator implemented in TinyGo." No dedicated PicoCalc target in the TinyGo BSP; uses the generic pico / pico2 targets and you bring your own LCD/keyboard code.

- https://tinygo.org
- Forum thread: https://forum.clockworkpi.com/t/hello-world-and-rpn-calculator-implemented-in-tinygo/20834

6.8 NuttX RTOS — port in progress

Active porting effort. SD/SPI/I²C/PWM/audio/UART working on RP2350; LCD + keyboard pending as of late-2025 forum activity. Watch this if you want a real preemptive RTOS instead of MicroPython's GIL-y world.

- Status thread: https://forum.clockworkpi.com/t/nuttx-rtos-on-picocalc/16783

6.9 FUZIX — mini-UNIX in 32 MB

Lightweight Unix-like OS. Ships in the default SD bundle and lives on the 32 MB raw partition. Boots a /bin/sh, has a small set of UNIX userland tools, runs in either RAM-only or with a writable rootfs on SD.

- UF2: PicoCalc_Fuzix_v1.0.uf2 in releases

6.10 Phyllosoma — compiled BASIC

Faster than PicoMite (compiled vs interpreted). Full keyboard support. Less famous, but worth a look if PicoMite's interpretive overhead bites a particular project.

- UF2: phyllosoma_kb.uf2 in the SD bundle

6.11 Picoware — MicroPython-based custom firmware

Higher-level "everything bundled" custom firmware: Wi-Fi tools, BLE, social-media clients, games. Useful baseline if you have a Pico 2 W and want to skip the integration step.

- Forum: https://forum.clockworkpi.com/t/picoware-open-source-custom-firmware/18536

6.12 What's missing

- **No JavaScript port** found that targets PicoCalc specifically. (mJS/Espruino runs on Pico generically; nobody has wrapped the LCD/keyboard.)
- **No CircuitPython** port found — Adafruit's CircuitPython runs on Pico 1/2/2W generically, but no PicoCalc display/keyboard drivers.
- **No native Linux on the standard Pico**. To run Linux on this case, swap the Pico module for a **Pi Zero 2 W carrier** (community calls this "PicoCalc Trixie" or "ZeroCalc"), or for a **LuckFox Lyra** SoM. Both fit the same socket. Treat as a separate project.

6.13 Stack picker (heuristic)

You want…	Pick
Most on-device productivity, zero	PicoMite

host computer	
Fastest iteration with host-side push	MicroPython (LofiFren)
Maximum performance, deterministic timing, PIO	Pico SDK C/C++
Lisp	uLisp (lock BIOS at 1.2)
Compiled-BASIC speed	Phyllosoma
A real RTOS	NuttX (when port lands)
Pocketable Linux	Swap the Pico for Pi Zero 2 W or LuckFox Lyra
Modern systems language	Rust (embassy) or TinyGo

7. Display & keyboard programming

The reason picocalc-text-starter is the canonical reference: it consolidates the per-peripheral drivers into a single picocalc.h umbrella that retargets newlib's stdio. Once picocalc_init(NULL) runs, printf/scanf go to the LCD + keyboard, and fopen/fwrite go to the SD card.

7.1 The starter's driver layout

```
picocalc-text-starter/
├── drivers/
│   ├── picocalc.h          # umbrella header
│   ├── lcd.c / lcd.h       # SPI display driver
│   ├── keyboard.c / .h     # I2C reader against STM32
│   ├── audio.c / .h        # PWM tone generation
│   ├── southbridge.c / .h  # STM32 power/backlight control
│   └── clib.c              # newlib retargets (stdio, file IO)
└── src/main.c              # demo REPL
```

7.2 Public API (sketch — exact signatures evolve)

```c
#include "picocalc.h"

int main(void) {
    picocalc_init(NULL);       // SPI, I2C, SD, audio, stdio retarget
    printf("\x1b[2J\x1b[H");    // ANSI clear screen + home cursor
    printf("Hello, PicoCalc!\n");
    int c = getchar();         // blocks, reads from keyboard
    printf("You pressed: %d\n", c);
    while (1) tight_loop_contents();
}
```

The terminal layer understands a useful subset of ANSI escape codes (cursor, color, clear). For graphics, drop down to lcd_* calls in lcd.h.

7.3 Keyboard event model

- STM32F103 scans the matrix and pushes key events into a small FIFO.
- Pico polls the STM32 over I^2C at 10 kHz bus speed (above 100 kHz drops bytes — confirmed by DeepWiki troubleshooting).
- Reads are request/response: write the "fetch key" register, read 2 bytes (keycode + modifier).
- Function keys (F1–F10), arrow/D-pad, ESC, BREAK each have ASCII-extended codes — exact map is in keyboard.c of text-starter or in Code/picocalc_keyboard/include/keymap.h of the official repo.
- Backlight (LCD + keyboard separately) is set via dedicated I^2C registers — not Pico GPIO.

7.4 Frame buffer strategies

- **RP2040 (Pico 1)**: 204,800 bytes is most of SRAM. Three options:
    a. **Tile/stripe**: keep an 8-row stripe in RAM, render scanlines into it, push, repeat. picocalc-text-starter does this for its text terminal.
    b. **PSRAM full-FB**: place the FB in the carrier's 8 MB PSRAM. Slower writes (~10–20 MB/s vs ~50 MB/s for SRAM) but full random access. Helpful for sprite-heavy graphics.
    c. **DMA chains**: build the line in SRAM, DMA it out over SPI, prepare next line while it transmits. Effective for video-rate updates if SPI clock is high enough.
- **RP2350 (Pico 2 / Pico 2 W)**: full 320×320×16 fits in SRAM with 300+ KB to spare. Just allocate it.

7.5 LVGL / GUI library

awesome-pico-calc lists an LVGL integration. LVGL gives you proper widgets, themes, animations — appropriate for menus, settings UIs, instrument-style displays. Heavier than rolling your own. The DeepWiki has a programming-guide section on LVGL on PicoCalc.

---

8. PIO state machines on PicoCalc

The RP2040/RP2350's PIO is the platform's superpower — 8 (RP2040) or 12 (RP2350) tiny state machines that run independently of the CPUs at up to the system clock rate. PicoCalc-specific applications:

- **Audio output via PIO PWM** (forum #18339) — workaround for the GP26/GP27 single-PWM-slice limitation. PIO time-multiplexes the two outputs.
- **Custom display drivers** — currently the LCD goes through the RP2040 SPI peripheral. PIO-driven SPI can hit higher clocks if you're willing to work for it.

- **PicoDVI / DispHSTX (RP2350)** — luke-wren's PicoDVI outputs HDMI from the RP2040 via PIO; on RP2350 the HSTX peripheral does it natively. Nobody's wired this to PicoCalc's side header yet, but the GPIOs are exposed.
- **Logic-analyzer style sampling** — PIO can capture pins at >100 MS/s and push to RAM via DMA. Useful for the "hand-held protocol sniffer" project category (§10).
- **Bit-banged custom RF modulation** — PIO can synthesize OOK/FSK/Manchester at audio-to-MHz rates, output to a GPIO routed to an external transmitter. The PicoCalc isn't a Flipper, but with an external CC1101 / nRF24 module on the side header, PIO + the keyboard + the screen makes a respectable signal-research handheld.

PIO assembler reference: pico-sdk/pio/, the pico-examples repo's PIO directory, and Van H