Evan McCord Morse

Shelburne, VT · (802) 490-9355 · ping@emorse.dev · emorse.dev · codeberg.org/e-mo

Embedded systems engineer focused on bare-metal firmware, low-power wireless sensor networks, and open hardware. Three years of funded research experience designing and shipping a complete IoT platform, from custom PCBs to a from-scratch firmware stack, with reliability, measured performance, and data provenance at the center.

EXPERIENCE

Embedded Research Software/Firmware Engineer

Vermont State University — NSF EPSCoR research grant · Randolph Center, VT · Feb 2023 – Sep 2026

Sole software and firmware engineer on a research team ("Wisdom") exploring open IoT technology for remote environmental sensing and infrastructure monitoring (bridges, culverts). Co-led redirection of the project from passive data collection toward novel open-source hardware and data-collection systems.

- Designed and built a solar-powered, low-power hub-and-spoke sensor network: battery-operated RP2040 sensor nodes that talk RFM69 packet radio to a cellular gateway (SIM7080G), which publishes to MQTT. Runs on sodium-ion cells with small solar panels and stays stable through multi-day stretches without sun.
- Achieved 100% data retention across deployed 5-node networks. When a gateway failed, nodes buffered readings to onboard EEPROM and shipped everything once the link was restored, exactly as designed.
- Wrote the entire firmware platform from scratch in Zig with no vendor SDK: startup code, linker scripts, boot stage 2, code-generated register definitions, a HAL covering 13 RP2040 peripherals, and 8 device drivers (radio, cellular modem, sensors, EEPROM, RTC, battery charger). About 129k lines of first-party code with 1,240+ unit tests across 17 repositories.
- Built a data-driven bare-metal build system. Hardware is described in declarative manifests, using a configuration format and parser toolkit I designed, and register-access code is generated from the manifests and wired directly into the build graph. 28 manifests cover the full RP2040 peripheral set.
- Designed RUDP, a reliable radio transport for lossy low-power links: fragmentation with selective repeat, duplicate suppression across reboots, and durable acknowledgments flushed to EEPROM before acking. Wakes a dormant node over the air in a measured 19 ms and delivers byte-for-byte at 40% injected bidirectional packet loss.
- Implemented over-the-air firmware updates over the RFM69 radio link, plus a host-side REPL and tooling for field diagnostics.
- Championed a fully open-source, open-hardware approach built around scientific data provenance: auditable hardware and software at every layer, with parts chosen so other researchers can reproduce the boards (hardware licensed CERN-OHL-W).
- Partnered weekly with the project's hardware engineer across 5 revisions of custom sensor-node PCBs (KiCad): design collaboration, board bring-up, firmware/hardware debugging, and fine-pitch soldering and rework.
- Documented everything to a professional standard: protocol and API references, engineering write-ups, and a measurements document where every timing constant names the benchmark firmware that produced it.

SELECTED PROJECTS

WHALE, an open firmware platform for the RP2040. The research platform above, released as independent open-source libraries: a parser-combinator toolkit, a typed configuration format, a register-code generator, a bare-metal build system, an RP2040 HAL and driver set, a USB device stack, a reliable radio transport with OTA update, and 26 documented example firmwares. Public release planned Sep 2026.

ECS performance study, modern Fortran vs. C. Implemented the core of an entity component system twice, in C and in modern Fortran, to test Fortran's claims of superior compiler auto-vectorization. Benchmarked hundreds of thousands of transformations over large data vectors, repeated thousands of times with randomized data. The Fortran implementation ran a consistent 10-15% faster than the C implementation. Wrote an accompanying paper.

limpet, a multiplexed IO monitor in Go. A tmux-style client/server tool that monitors many serial (USB CDC) and MQTT streams at once, with auto-reconnect, a uniform stream-naming scheme, and pluggable formatters. Built as daily-driver tooling for the sensor-network work. About 12k lines, with tests.

SKILLS

Languages: C, Zig, Fortran (modern), Go, Python, Rust, C++, Java, Kotlin, SQL. Comfortable picking up any language quickly.
Embedded: bare-metal firmware, RP2040, register-level peripheral programming (SPI, I2C, UART, USB, ADC, DMA, clocks/PLL), low-power design, JTAG/SWD debugging, OTA update, RF (packet radio, cellular/LTE-M), MQTT
Hardware: component selection and datasheet-driven design, KiCad (schematics, design review), board bring-up, oscilloscopes, signal generators, bench power supplies, fine-pitch soldering and rework
Tools & practice: Linux (daily driver 10+ years), Git (advanced workflows), terminal-centric development, build systems, unit testing at scale, technical writing
AI-assisted development: experienced pairing with frontier LLMs and building custom harnesses. A disciplined workflow that accelerates output without ceding engineering judgment.

EDUCATION

B.S. Computer Engineering — Vermont State University (formerly Vermont Technical College), Randolph Center, VT · May 2026
President's List × 3 semesters, Dean's List × 7 semesters

A.S. Information Technology — Community College of Vermont

HONORS

Outstanding Student Presentation Award, American Geophysical Union (AGU) Fall Meeting 2025. Awarded for a MacGyver-session poster defense of the sensor-network research ("Think Differently: Exploring Non-Proprietary Hardware Solution for Remote Data Collection"), selected from thousands of student presentations at the world's largest Earth and space science conference (~25,000 attendees).