OPENVVVF

SOFTWARE PLAN

# Codebase Improvement Plan

Readability, reliability, and safety hardening plan for the RTE Studio and firmware codebase.

| | |
|---|---|
| **Document ID** | OV-SW-RTE-IMPROVEMENT |
| **Version** | 1.0 |
| **Date** | 2026-07-13 |
| **Status** | draft |
| **Applies to** | software-target-rte-studio |
| **Product line** | openvvvf |

# 1. Codebase Improvement Plan - Readability, Reliability, Safety

## 1.1. Goals

- Make the firmware **harder to misuse** and easier to review.
- Close the known safety gaps from `reminder.md` (watchdog, overcurrent fault integration).
- Reduce the class of bugs that come from HAL errors, silent telemetry drops, ISR/main-loop races, and unvalidated hardware state restores.
- Make the code **testable on the host** for pure-logic pieces.
- Keep the existing, verified current-sensor offset calibration sequence intact.

## 1.2. Guiding principles

- **Safety first:** every change must either improve a safety path or be neutral to it.
- **Minimal disruption:** refactor in small, reviewable steps; do not re-architect working hardware sequences unless required.
- **Fail loudly:** turn ignored return values, timeouts, and precondition violations into logged faults or assertions.
- **Single source of truth:** constants live in one place; hardware ownership is explicit.

## 1.3. Phase 1 - Safety-critical hardening

### 1.3.1. Independent watchdog (IWDG)

- Enable `HAL_IWDG_MODULE_ENABLED` in `Inc/stm32h7xx_hal_conf.h`.
- Add a `Watchdog` module in `Src/Inverter/Control/Watchdog.cpp`.
- Configures IWDG with a timeout appropriate for the main-loop rate (~10–50 ms).
- Exposes `feed()` called from `InverterMain::loop()`.
- Adds a `WatchdogMonitor` that tracks the last-feed time and raises `FaultSource::SupplyVosrdy` (or a new `WatchdogTimeout`) if the loop starves.
- Acceptance: pulling the debugger halt for longer than the timeout resets the board.

### 1.3.2. Software overcurrent → latched Critical fault

`PhaseCurrentADC` currently calls `FaultManager::raise(PhaseOvercurrent)` only when an explicit threshold is set, and the default is 1000 A. Make it a first-class safety feature:

- In `Inc/Inverter/Drivers/Sensors/PhaseCurrentADC.h` add a configurable default (e.g. 150 A) and separate thresholds for calibration vs. run modes.
- In `Src/Inverter/Drivers/Sensors/PhaseCurrentADC.cpp`:
- Always run the overcurrent check in the ISR.
- Raise `FaultSource::PhaseOvercurrent` with reason `PhaseOvercurrentSoftware` when the magnitude on **any** phase exceeds the threshold.
- Add telemetry keys `ph_oc_threshold_a`, `ph_oc_max_a`.
- Add shell commands in `Src/Inverter/Command/Commands/SensorCommands.cpp` to set the threshold with a hard upper bound (e.g. 500 A) to prevent accidental disable.
- Acceptance: `fault_test PhaseOvercurrent` still works; a real overcurrent trips `FaultManager::executeSafetyActions()`.

### 1.3.3. Centralized safety monitor

Create `Src/Inverter/Control/SafetyMonitor.cpp` and call it from `InverterMain::loop()` right before `executeSafetyActions()`.

It checks and faults on:

- PWM outputs enabled while a Critical fault is active.
- Gate-driver power enabled but `/RDY` low for >100 ms outside calibration.
- Current-sensor offset invalid (`!phaseCurrentADC().offsetValid()`).
- Encoder samples not arriving (`encoderADC()` sample timeout) - re-enable the disabled fault from `EncoderADC::diagnose()`.
- Main-loop period exceeded (record `loop_dt_ms` and alarm if >20 ms).

Each check gets a `SafetyCheck` enum, a counter, and a configurable trip threshold. Keep it simple and explicit.

### 1.3.4. Correct interrupt save/restore

Replace `__disable_irq()` / `__enable_irq()` pairs that are called from both ISR and thread context with a PRIMASK save/restore helper:

- Add `Inc/Inverter/Control/CriticalSection.h` with `struct CriticalSection { uint32_t primask; enter(); leave(); }`.
- Apply it in:
- `FaultManager::raise/clear/activeFlags`
- `PhaseCurrentADC::sample`
- `EncoderADC::sample/resetBounds/learnBounds`
- `Telemetry::uart_write_bytes/onUartTxComplete`

- This prevents accidentally enabling interrupts inside an ISR.

### 1.3.5. Assertion / invariant macros

Add `Inc/Inverter/Control/DebugAssert.h`:

```
#define INVARIANT(cond, msg) \
    do { if (!(cond)) { Telemetry::printf("ASSERT: %s", msg); \
                        FaultManager::instance().raise(FaultSource::AdcError,
FaultReason::AdcHalError); \
                        __BKPT(0); } } while(0)
```

Use it for:

- State-machine preconditions (e.g. never enter `MEASURE` from `IDLE`).
- Division-by-zero guards before fits in `ResistanceCalibrator`.
- Valid parameter ranges in `start()` methods.

For release builds the macro can degrade to a fault raise only.

## 1.4. Phase 2 - Reliability improvements

### 1.4.1. Propagate HAL errors and fault on init failures

- `PhaseCurrentADC::init/start`, `EncoderADC::init/start`, `MAX22530::init`, `DcLinkVoltageSensor::init` currently ignore or silently continue on HAL errors.
- Change them to return `bool`, log a descriptive string, and raise the matching `FaultSource` / `FaultReason` on failure.
- In `InverterMain::init()` halt startup with a visible error if any safety-critical driver fails (or at least latch the fault so the shell can report it).

### 1.4.2. Telemetry overflow / drop detection

- In `Src/Inverter/Telemetry.cpp` count dropped defines, dropped logs, and UART ring-buffer overflows.
- Publish telemetry keys `tlm_drops_define`, `tlm_drops_log`, `tlm_tx_overflows`.
- If drops exceed a threshold in a window, raise `FaultSource::UartError` so the host knows telemetry is unreliable.

### 1.4.3. Fix the duplicated `TIM1->BDTR` restore in `ResistanceCalibrator`

In `Src/Inverter/Calibration/ResistanceCalibrator.cpp` `restoreHardware()` does:

```
TIM1->BDTR = (m_saved_bdtr & ~TIM_BDTR_DTG) | (TIM1->BDTR & TIM_BDTR_DTG);
TIM1->BDTR = m_saved_bdtr;
```

The second write overwrites the first. Keep only the second write **after** the timer is stopped and outputs are safe, or restore the dead-time field carefully. Add a comment explaining the restore order.

### 1.4.4. Validate saved/restored hardware state

Before trusting `m_saved_*` registers, verify they are within sane ranges:

- `ARR` in `[1000, 65535]`.
- `PSC` small.
- `CCER` only has expected phase-channel bits set.
- If validation fails, fall back to a known-safe default (10 kHz, 50 % duty, gate driver reset).

This prevents a bad save from corrupting the next run.

### 1.4.5. Add timeouts to busy-wait calibration loops

`PhaseCurrentADC::calibrateOffsets()` spins forever on `m_new_data`. Add a max-wait counter (e.g. 10 ms) and fail if no sample arrives, raising a fault.

### 1.4.6. Unify gate-driver ownership

`OpenLoopController`, `CalibrationHardware`, `ResistanceCalibrator`, and `EncoderOffsetCalibrator` all touch the gate-driver reset line. Introduce a single `GateDriverState` enum owned by `OpenLoopController`:

- `OFF` (power off)
- `RESET` (power on, reset asserted)
- `READY` (reset released, /RDY high, /FLT low)

Other modules request states through `OpenLoopController::requestGateDriverState()` instead of writing GPIO directly. This eliminates races where one module asserts reset while another thinks outputs are enabled.

### 1.4.7. Command parser hardening

`CommandManager::processLine` uses fixed 32-byte token buffers and `atof` / `atoi`. Improve:

- Increase token size to 64 bytes for long floats/strings.
- Report truncation instead of silently cutting input.
- Add a `"help <command>"` mode that prints per-command usage.

### 1.4.8. Re-enable encoder timeout fault

Remove the commented-out code in `EncoderADC::diagnose()` and make the timeout configurable (e.g. 200 ms). If it is too noisy during startup, only enable it after `offsetValid()` is true.

## 1.5. Phase 3 - Readability and maintainability

### 1.5.1. Eliminate magic numbers

Move hard-coded constants into named `constexpr`s near the top of their modules:

- `EncoderOffsetCalibrator.cpp`: `10000U` range checks, `45.0f` sign-detect threshold, `0.02f` noise threshold.
- `ResistanceCalibrator.cpp`: `1000U` settle/measure times, `8.0f` kHz comment should be a named `CAL_SWITCHING_FREQ_HZ`, `0.05f` PI constants.
- `PoleCalibrator.cpp`: `1.0f` target cycles, `0.5f` partial cycles, `120000U` max count.
- `SupplyMonitor.cpp`: PVD/AVD levels already named; document why.

### 1.5.2. Abstract the phase-pair mapping

In `ResistanceCalibrator` the active/inactive/high-Z phase mapping is duplicated in switches in `configureHardware()` and free functions `pairCurrentActive/Inactive`. Create a small `PhasePair` value type:

```
struct PhasePair {
    int high;   // 0=U, 1=V, 2=W
    int low;
    int high_z;
};
```

Add a function `PhasePair mapPair(Pair p)` and use it everywhere. This removes a source of copy-paste bugs.

### 1.5.3. Split oversized modules

- `ResistanceCalibrator.cpp` (837 lines) →
- `ResistanceHardware.cpp` (TIM/GPIO setup, save/restore)
- `ResistanceFitter.cpp` (linear regression, inactive-current checks)
- `ResistanceCalibrator.cpp` (state machine only)
- `Telemetry.cpp` (833 lines) →
- `TelemetryProtocol.cpp` (COBS, CRC, frame builder)

- `TelemetryTransport.cpp` (UART DMA ring)
- `Telemetry.cpp` (public API + queues)
- Keep header interfaces stable to avoid churn.

### 1.5.4. State-machine helper

Add a small `StateMachine<State>` helper that records:

- current state
- entry time
- allowed transitions ( `isTransitionAllowed(from, to)` )
- time-in-state

Use it in `ResistanceCalibrator` , `EncoderOffsetCalibrator` , `AutoCalibrationCoordinator` , and `OpenLoopController::StartupState` . Catches illegal transitions at runtime and logs them.

### 1.5.5. Remove dead / commented-out code

- Delete the old blocking ramp comments if the non-blocking ramp is final.
- Delete the disabled encoder timeout block or turn it into a runtime flag.
- Remove stale flash scripts ( `build_flash.sh` , `build_flash_uart_manual.sh` , `flash_uart_manual.py` ) or move them to an `archive/` directory.

### 1.5.6. Consistent formatting

- Add a `.clang-format` file (LLVM style with 4-space indent, 120 column limit).
- Run it on `Src/Inverter/**` and `Inc/Inverter/**` in a single formatting commit so future diffs are clean.
- Add `-Wall -Wextra -Werror` to CMake after the warning cleanup is done.

## 1.6. Phase 4 - Testing and static analysis

### 1.6.1. Host-side unit tests for pure logic

Add a `tests/` directory with a native CMake target that compiles on x86:

- `test_breakaway_finder.cpp`
- `test_current_limited_ramp.cpp`
- `test_encoder_tracker.cpp`
- `test_telemetry_encoding.cpp` (CRC, COBS, ring queue)
- `test_phase_pair.cpp`

- `test_fault_manager.cpp` (mock `Telemetry`)

Use **doctest** or **Catch2** (single-header, easy to vendor). Provide a mock `HAL_GetTick()` so state machines can be stepped deterministically.

### 1.6.2. Static analysis

- Add a `.clang-tidy` file with:
- `bugprone-*, cppcoreguidelines-pro-type-*, cppcoreguidelines-avoid-magic-numbers, clang-analyzer-*, performance-*, readability-*, modernize-*`
- Run `clang-tidy` against `compile_commands.json` and fix the highest-priority findings.
- Add `cppcheck` to the build for a second opinion.

### 1.6.3. Runtime diagnostics

- Add a `diag` shell command that prints:
- watchdog status
- loop timing stats (min/mean/max dt)
- telemetry queue depth and drop counts
- current-sensor offset validity
- gate-driver state
- active faults

This gives you one command to verify system health before high-power tests.

## 1.7. Implementation order

Recommended order, one PR/commit at a time:

1. Critical section helper + replace `__disable_irq` / `__enable_irq`.
2. Assertion/invariant macros + precondition checks in `start()` methods.
3. Watchdog module + feed in main loop.
4. Software overcurrent integration + default threshold.
5. Safety monitor.
6. Telemetry drop counters.
7. Fix `ResistanceCalibrator::restoreHardware()` BDTR double-write + saved-state validation.
8. Phase-pair abstraction + split `ResistanceCalibrator`.
9. Unify gate-driver state machine.
10. HAL error propagation in init paths.

11. Magic-number cleanup + `.clang-format` pass.
12. State-machine helper.
13. Host unit tests.
14. Static analysis cleanup + `-Werror`.

## 1.8. Acceptance criteria

- `motorcal` still succeeds three times in a row without rebooting.
- `fault_test PhaseOvercurrent` triggers the safety shutdown (PWM break, gate-driver reset, power off).
- Halting the debugger for >100 ms causes a watchdog reset.
- Telemetry reports zero drops during normal operation.
- `clang-tidy` runs with no `bugprone-*` or `clang-analyzer-*` findings.
- Host tests pass: `cmake --build build_tests && ctest`.

## 1.9. What to leave alone

- The verified current-sensor offset calibration sequence documented in `reminder.md`.
- The encoder angle computation and bound-learning logic in `EncoderADC`.
- The DC-link voltage scaling (1516.0) and TIM1 clock assumptions from `reminder.md`.