# Red Bird Racing EVRT Software Training Project

# Technical Handbook

**Author**: Arnav

**Date**: January 22, 2026

## System Architecture

The Vehicle Control Unit (VCU) is implemented on an ATmega328P microcontroller using the Arduino framework. It interfaces with:

- Analog sensors: Two Accelerator Pedal Position Sensors (APPS at 5V and 3.3V references) for throttle input, and a brake sensor.
- Digital input: Start button for initiating the drive sequence.
- Digital outputs: Brake light, buzzer for ready-to-drive sound, and drive LED indicator.
- CAN communication: Three MCP2515 controllers (20 MHz crystals) for separate buses.
    - Motor CAN (CS pin PB2 / 10): Transmits torque requests.
    - BMS CAN (CS pin PB1 / 9): Receives permission message from Battery Management System.
    - Debug CAN (CS pin PB0 / 8): Transmits telemetry for monitoring.

The system uses a finite state machine to manage vehicle modes with safety interlocks (brake requirement, APPS plausibility check, BMS permission).

## Logic Flow

The program runs in a continuous loop:

1. Read sensor values (APPS5V, APPS3V3, brake) and start button state.
2. Update brake light based on threshold.
3. Check for BMS permission message (ID 0x186040F3, byte 6 = 0x50).
4. Execute state machine:
    - INIT: Wait for start button + brakes depressed → transition to STARTIN.
    - STARTIN: Hold for 2 seconds, require BMS ready → transition to BUZZIN.
    - BUZZIN: Activate buzzer for 2 seconds → transition to DRIVE.

    - DRIVE: Calculate normalized APPS values, check for fault (difference >10% for >100 ms → reset to INIT), compute average pedal, generate torque, send torque CAN message.
5. Every 50 ms, send debug telemetry CAN message with current data.

The flow ensures safety: no torque without proper startup and continuous plausibility.

# CAN Message Interpretation

- **Torque Command (ID 0x201, DLC 3)**:
    - Byte 0: Always 0x90 (command identifier).
    - Bytes 1–2: Torque value (signed 16-bit, little-endian, -32768 to 32767).
    - Interpretation: Sent in DRIVE state; motor controller uses this for power output. Use SavvyCAN with my_vcu.dbc for decoding.
- **Debug Telemetry (ID 0x500, DLC 8)**:
    - Bytes 0–1: APPS5V raw (little-endian, 0–1023).
    - Bytes 2–3: APPS3V3 raw (little-endian, 0–675).
    - Byte 4: Vehicle state (0=INIT, 1=STARTIN, 2=BUZZIN, 3=DRIVE).
    - Bytes 5–6: APPS difference raw (little-endian, scaled to 5V range).
    - Byte 7: Fault active (0x01 = yes, 0x00 = no).
    - Interpretation: Sent every 50 ms for monitoring. Load my_vcu.dbc in SavvyCAN to see decoded values like "VehicleState = DRIVE".

# Maintenance Notes

- **Adjust safety parameters**: Change APPS_FAULTY_THRESHOLD (102) for different plausibility sensitivity; test thoroughly to avoid false faults.
- **Timing changes**: Modify STARTIN_HOLD_TIME or BUZZIN_TIME for different startup durations.
- **Debugging**: Use SavvyCAN with my_vcu.dbc to monitor CAN; add Serial prints for non-CAN debugging if needed.
- **Library updates**: If upgrading MCP2515.h, check API compatibility (e.g. sendMessage).
- **Hardware notes**: Ensure 20 MHz crystals; if CAN fails, check SPI wiring and CS pins.
- **Extensions**: Add more debug signals (e.g. brake raw in unused bytes) for BONUS.