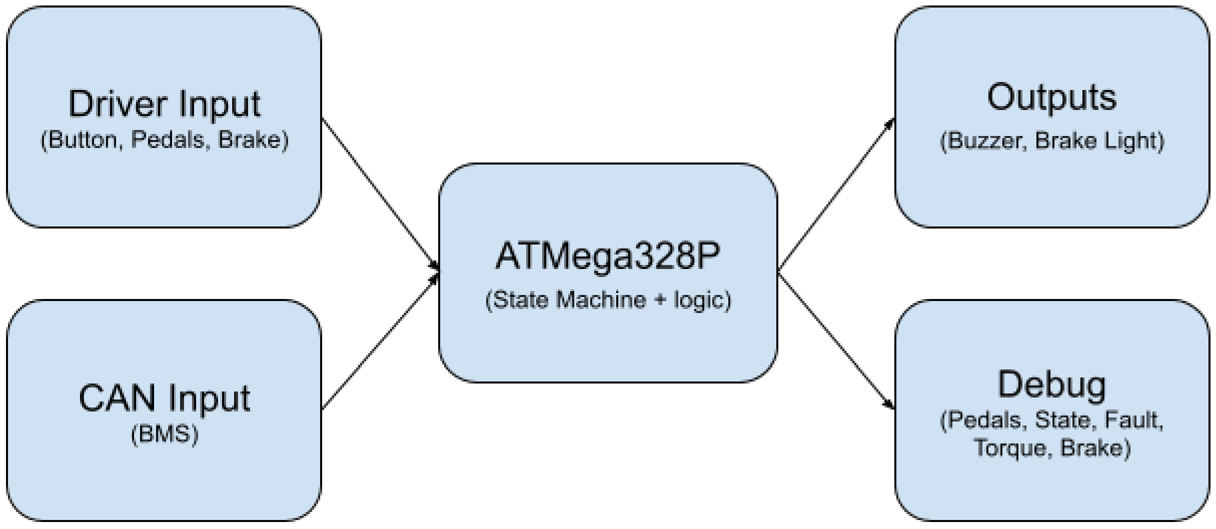
VCU Technical Handbook

System Architecture

The VCU is an embedded control system built on an ATMega328P microcontroller using the Arduino framework. It interfaces with driver inputs, and vehicle components, such as motor and brake light, via analog/digital pins and three separate CAN buses.

- **Hardware:**
    - **Microcontroller:** ATMega328P (handles state machine, input reading, and logic).
    - **CAN Controllers:** Three MCP2515 modules (500 kbps, 20 MHz):
        - Motor bus (CS: PB2) – Sends torque commands.
        - BMS bus (CS: PB1) – Receives signal from BMS to perform state changing.
        - Debug bus (CS: PB0) – Sends messages for monitoring.
    - **Inputs:**
        - APPS_5V (PC0) – Motor torque.
        - APPS_3V3 (PC1) – Motor torque.
        - Brake (PC3) – Brake pressure.
        - Start button (PC4) – Starting the vehicle.
    - **Outputs:**
        - Digital: Brake light (PD2), Drive LED (PD3), Buzzer (PD4).
    - **Power/Safety:** System assumes 5V/3.3V scaling for APPS, with fault detection to prevent torque output during pedal failure.

The system uses a state machine to ensure safe vehicle startup and operation. Inputs feed into the MCU, which processes logic and outputs via pins or CAN. Debug CAN allows real-time monitoring without interfering with critical buses.



Rough Logic Flow of the Program

The program runs in a continuous loop with a 10 ms delay, implementing a state machine for control. Key globals include thresholds (APPS disagreement 102, brake depressed 600) and timings (STARTIN/BUZZIN 2000 ms).

- **Setup Phase:**
    1. Configure pins: Inputs for pedals/brake/button, outputs for light/LED/buzzer (initially LOW).
    2. Initialize MCP2515 CAN controllers: Reset, set bitrate 500 kbps, normal mode.
    3. Set initial state to INIT with current millis timestamp.
- **Main Loop (every 10 ms):**
    1. Read inputs: analogRead for APPS5V/3V3/brakeRaw; digitalRead for startPressed.
    2. Update brake status: brakeDepressed if >=600, brakeLightOn if >=102; set light pin HIGH/LOW.
    3. Send debug message (0x300): Pack APPS readings, state, fault, diff (only if faulty).
    4. Switch on vcuState:
        - **INIT:** Torque 0, outputs off. If startPressed && brakeDepressed, transition to STARTIN, update timestamp.
        - **STARTIN:** Torque 0. If !startPressed || !brakeDepressed, back to INIT. Check BMS CAN receive (ID 0x186040F3, data[6]==0x50) to go BUZZIN.
        - **BUZZIN:** Torque 0, buzzer HIGH. After 2000 ms, buzzer LOW, go DRIVE, LED HIGH.
        - **DRIVE:** LED HIGH. Check APPS fault (scaled diff >102 or <-102). If faulty, start timer; if >100 ms, torque 0, LED off, back INIT. Else, convert APPS5V to torque (shift <<5, optional reverse), send torque (0x201 DLC=3 + debug 0x400 DLC=6 with brake).
    5. Delay 10 ms.

Flow safety: Faults or start button release will always reset to INIT with zero torque.

How to Interpret the CAN Messages

Use SavvyCANwith the provided debug.dbc file to decode messages.

- **Motor Torque Command (ID 0x201 / 513, DLC 3):**
  - Byte 0: 0x90 (fixed command ID).
  - Bytes 1-2: Torque (-32768-32767, signed 16-bit).
- **Debug Message (ID 0x300 / 768, DLC 8):**
  - Bytes 0-1: APPS5V raw ADC (0-1023, unsigned).
  - Bytes 2-3: APPS3V3 raw ADC (0-1023, unsigned).
  - Byte 4: State (1="INIT", 2="STARTIN", 3="BUZZIN", 4="DRIVE").
  - Byte 5: Fault (0="OK", 1="FAULT" if APPS diff >102).
  - Bytes 6-7: Difference between 5V, 3V3 (-32768-32767, signed 16-bit, meaningful only if fault=1, else 0).
- **Torque/Brake Debug (ID 0x400 / 1024, DLC 6):**
  - Byte 0: 0x90 (command ID, neglected while decoding).
  - Bytes 1-2: Torque (signed 16-bit, same as motor).
  - Bytes 3-4: BrakeRaw ADC (0-1023, unsigned).
  - Byte 5: Brake_Light (0="Off", 1="On" if brakeRaw >=102).
- **BMS Receive (ID 0x186040F3, DLC >=7):**
  - Data[6] == 0x50: Triggers STARTIN to BUZZIN.

In SavvyCAN, load debug.dbc, enable Interpret Frames for human-readable decoding (e.g., state as "DRIVE", fault as "OK").

Maintenance

- **Tuning Thresholds:** Adjust constants like APPS_MAX_DISAGREE_THRESHOLD (102) or BRAKE_DEPRESSED_THRESHOLD (600) based on real hardware optimisation. Test pedal under noise, use debug CAN logs in SavvyCAN to analyze diffs and set safer thresholds.
- **Updating Code:** Keep changes isolated (e.g., new features in helpers like convertMotorTorque). Always verify state transitions with sim logs before hardware flash. Backup GitHub repo versions.
- **Hardware Swap:** Pins in #defines; if changing (e.g., new CS pin), update and re-init MCP. Monitor for CAN errors (use mcp2515.readMessage return codes).
- **Future Developments(for bonus):** Add torque curve in convertMotorTorque.