ECU design for fuel indicator

Conceptual Hardware Design ⛽

The system for measuring fuel level is straightforward and robust.

- **The Sensor (Fuel Sending Unit):** Inside the fuel tank, there is a **float** attached to an arm. This arm moves the wiper of a **variable resistor** (potentiometer).
    - As the fuel level changes, the float moves, changing the sensor's resistance.
    - Typically, the resistance is low when the tank is **full** and high when it's **empty**.
- **Voltage Divider Circuit:** The variable resistor from the sending unit is placed in a **voltage divider** circuit with a known, fixed resistor. The ECU measures the voltage at the midpoint of this circuit.
- **ECU (Microcontroller with ADC):** The ECU's **Analog-to-Digital Converter (ADC)** reads the voltage from the circuit. This voltage directly corresponds to the fuel level.
- **Output (Display):** The processed information is sent over the **CAN bus** to the instrument cluster, which then updates the physical fuel gauge needle or a digital display.

Key Challenge: Fuel Sloshing Vehicle movement causes fuel to slosh around in the tank, which would make a raw sensor reading fluctuate wildly. The primary job of the ECU's software is to filter these readings to provide a stable display.

Enhancements for a Real-World System 🛠️

- **Non-Linear Tank Mapping:** Most fuel tanks have irregular shapes. A real ECU uses a multi-point **lookup table** (not a simple linear formula) to accurately map the sensor's resistance to the actual fuel volume.
- **Low Fuel Warning:** The ECU is responsible for triggering the low fuel warning light when the calculated percentage drops below a specific threshold (e.g., 12%).
- **Sensor Diagnostics:** The code would include checks to see if the ADC reading is outside its plausible range (e.g., near 0 or 1023), which would indicate a short or open circuit in the sensor wiring. This would log a fault code (DTC).
- **Intelligent Filtering:** The filtering logic could be made smarter. For example, the averaging period might be made longer when the vehicle is detected to be moving (via CAN bus speed data) and shorter when it's stationary for a quicker update after refueling.

```
#include <stdint.h>
#include <stdio.h> // For printf simulation
#define ADC_CHANNEL_FUEL    1
// --- Calibration values (found experimentally) ---
// Lower ADC value means higher voltage, lower resistance -> FULL tank
#define ADC_VAL_FULL       150
// Higher ADC value means lower voltage, higher resistance -> EMPTY tank
#define ADC_VAL_EMPTY      900

// --- Software Filter Definitions ---
#define FILTER_SAMPLES     50 // Number of samples to average

static uint16_t adc_buffer[FILTER_SAMPLES] = {0};
static uint8_t  buffer_index = 0;
static uint32_t running_sum = 0;
// Placeholder for reading a raw value from the ADC
uint16_t read_adc_value(uint8_t channel) {
   // This function would trigger the ADC hardware to perform a conversion.
   // We can simulate a reading for demonstration.
   // Let's simulate a value for a tank that is about 75% full.
   return 338;
}

/**
 * @brief Updates a moving average filter with a new raw ADC value.
 * @param new_adc_val The latest raw reading from the sensor.
 * @return The filtered (averaged) ADC value.
 */
uint16_t update_fuel_filter(uint16_t new_adc_val) {
   // Subtract the oldest value from the sum
   running_sum -= adc_buffer[buffer_index];
```

```
    // Add the new value to the buffer and the sum
    adc_buffer[buffer_index] = new_adc_val;
    running_sum += new_adc_val;

    // Move to the next position in the buffer
    buffer_index++;
    if (buffer_index >= FILTER_SAMPLES) {
        buffer_index = 0;
    }

    // Return the average
    return (uint16_t)(running_sum / FILTER_SAMPLES);
}
int main(void) {
    // In a real MCU, initialize peripherals.
    // Prime the filter buffer with initial readings for a stable start.
    uint16_t initial_val = read_adc_value(ADC_CHANNEL_FUEL);
    for (int i = 0; i < FILTER_SAMPLES; i++) {
        update_fuel_filter(initial_val);
    }
    printf("Filter initialized. Starting main loop.\n");

    // Main Superloop
    while(1) {
        // 1. Get a new raw reading from the sensor
        uint16_t raw_adc = read_adc_value(ADC_CHANNEL_FUEL);

        // 2. Pass it through the filter to get a stable, averaged value
        uint16_t filtered_adc = update_fuel_filter(raw_adc);
```

```
        // 3. Map the stable ADC value to a percentage
        // Clamp the values to the calibrated range to avoid over/under-flow
        if (filtered_adc < ADC_VAL_FULL)  filtered_adc = ADC_VAL_FULL;
        if (filtered_adc > ADC_VAL_EMPTY) filtered_adc = ADC_VAL_EMPTY;

        // Linear conversion from ADC range to percentage
        float fuel_percent = 100.0 * (float)(ADC_VAL_EMPTY - filtered_adc) / (float)(ADC_VAL_EMPTY - ADC_VAL_FULL);

        printf("Raw ADC: %-4d -> Filtered ADC: %-4d -> Fuel Level: %.1f%%\n", raw_adc, filtered_adc, fuel_percent);

        // 4. In a real car, this percentage would be sent over the CAN bus.
        // can_send_fuel_level(fuel_percent);

        // Wait a short period before the next reading
        // delay_ms(200);
    }

    return 0;
}
```