ECU design for measurement of heat produced in engine

To measure engine temperature, an ECU relies on a simple and robust set of components.

- **The Sensor (NTC Thermistor):** The core of the system is a sensor, almost always an **NTC (Negative Temperature Coefficient) thermistor**. This is a special type of resistor whose resistance drops predictably as its temperature rises. It's screwed directly into the engine block where it can contact the engine coolant.
- **Voltage Divider Circuit:** A microcontroller cannot directly measure resistance. Instead, we use a **voltage divider**. The NTC thermistor is connected in series with a known, fixed resistor. A stable 5V is applied across the pair. As the engine heats up, the thermistor's resistance drops, causing the voltage at the midpoint of the circuit to change.
- **Microcontroller with ADC:** The ECU's microcontroller uses an **ADC (Analog-to-Digital Converter)** pin to read the analog voltage from the midpoint of the voltage divider. The ADC converts this voltage into a digital number.
- **Output / Actuator:** Based on the calculated temperature, the ECU performs critical actions:
    - **Cooling Fan Relay:** It can turn the engine's cooling fan on or off.
    - **CAN Bus:** It broadcasts the temperature data to the instrument cluster for the driver's gauge and to other ECUs that need this information.

Signal Flow: Engine heats up → NTC thermistor resistance drops → Voltage at divider changes → Microcontroller ADC reads voltage → ECU logic converts voltage to temperature → ECU controls cooling fan.

Enhancements for a Real-World System 🛠

- **Sensor Diagnostics:** A production ECU would constantly check if the sensor reading is rational. An extremely high resistance (open circuit) or extremely low resistance (short to ground) would trigger a **Diagnostic Trouble Code (DTC)** and activate a failsafe strategy.
- **Failsafe Strategy:** If the sensor fails, the ECU can't know the real temperature. To prevent engine damage, it will assume the worst-case scenario (engine is overheating) and command the cooling fan to run continuously.
- **Signal Filtering:** To avoid reacting to minor electrical noise, the ADC values are typically passed through a software low-pass filter (like a simple moving average) to get a stable reading.
- **CAN Bus Communication:** Instead of just printing the value, the ECU would package the temperature data into a **CAN message** and broadcast it on the vehicle network for the instrument cluster and other modules.

```
#include <stdint.h>
#include <stdio.h> // For printf simulation
#define ADC_CHANNEL        0      // ADC channel connected to the voltage divider
#define ADC_MAX_VAL        1023.0 // For a 10-bit ADC
#define VOLTAGE_REF        5.0    // Reference voltage in Volts
#define FIXED_RESISTOR     10000.0// Value of the fixed resistor in Ohms
#define COOLING_FAN_PIN    8      // Output pin to control the fan relay
// --- Hysteresis thresholds for fan control ---
#define FAN_ON_TEMP_C      95     // Turn fan ON if temp exceeds 95°C
* SENSOR LOOKUP TABLE (LUT)
 * Resistance (Ohms) vs. Temperature (°C) for a typical NTC thermistor.
 * NOTE: In a real ECU, this table would be much larger.
 *
============================================================================*/
const float resistance_table[] = { 34000, 17000, 9300, 5300, 3180, 1990, 1290, 840, 325, 177 };
const int   temp_table[]      = { -20,    0,     20,   40,   60,   80,   100,  120, 160, 200 };
const int   TABLE_SIZE        = 10;
// Placeholder for reading a raw value from the ADC
uint16_t read_adc_value(uint8_t channel) {
   // This function would configure and trigger the ADC hardware.
   // For simulation, let's return a sample value.
   // A value of 325 corresponds to ~1.59V, which is around 95°C with this circuit.
   return 325;
}

// Placeholder for controlling a GPIO pin
void set_pin_state(uint8_t pin, bool state) {
   printf("Pin %d set to %s\n", pin, state ? "HIGH (ON)" : "LOW (OFF)");
}
```

```
 * @brief Converts a raw ADC value to an engine temperature in Celsius.
 * @param adc_val The raw digital value from the ADC (0-1023).
 * @return The calculated temperature in degrees Celsius.
 */
float adc_to_temperature(uint16_t adc_val) {
    // 1. Convert ADC value to voltage
    float voltage_out = (adc_val / ADC_MAX_VAL) * VOLTAGE_REF;
    // 2. Calculate the thermistor's resistance using the voltage divider formula:
    // R_thermistor = R_fixed * (V_ref / V_out - 1)
    float resistance = FIXED_RESISTOR * (VOLTAGE_REF / voltage_out - 1.0);
    // 3. Convert resistance to temperature using the Lookup Table and linear interpolation
    // Find the two resistance points our value falls between
    int i;
    for (i = 0; i < TABLE_SIZE; i++) {
        if (resistance > resistance_table[i]) {
            break;
        }
    }
    // Handle edge cases (out of range)
    if (i == 0) return temp_table[0];
    if (i == TABLE_SIZE) return temp_table[TABLE_SIZE - 1];
    // 4. Perform linear interpolation
    // y = y1 + (x - x1) * (y2 - y1) / (x2 - x1)
    float r1 = resistance_table[i-1];
    float t1 = temp_table[i-1];
    float r2 = resistance_table[i];
    float t2 = temp_table[i];
    float temperature = t1 + (resistance - r1) * (t2 - t1) / (r2 - r1);

    return temperature;
}
```

```
int main(void) {
    // In a real MCU, we would initialize GPIO, ADC, and other peripherals here.

    static bool is_fan_on = false;

    // Main Superloop
    while(1) {
        // --- Measurement Step ---
        uint16_t raw_adc = read_adc_value(ADC_CHANNEL);
        float current_temp = adc_to_temperature(raw_adc);

        printf("Raw ADC: %d -> Calculated Temp: %.2f°C\n", raw_adc, current_temp);

        // --- Control Step (with Hysteresis) ---
        if (!is_fan_on && current_temp > FAN_ON_TEMP_C) {
            printf("ACTION: Temperature high! Turning fan ON.\n");
            set_pin_state(COOLING_FAN_PIN, true);
            is_fan_on = true;
        } else if (is_fan_on && current_temp < FAN_OFF_TEMP_C) {
            printf("ACTION: Temperature normal. Turning fan OFF.\n");
            set_pin_state(COOLING_FAN_PIN, false);
            is_fan_on = false;
        }

        // Delay for a short period before the next reading
        // delay_ms(500);
    }

    return 0;
}
```