Session 14 - A ECU design for dash board panel

ECU Design for Dashboard Panel (Instrument Cluster)

I. High-Level Requirements:

1. **Display Basic Vehicle Status:**
    - **Vehicle Speed**
    - **Engine RPM**
    - **Fuel Level**
    - **Engine Temperature**
2. **Warning/Indicator Lights:**
    - **Check Engine Light**
    - **Low Fuel Warning**
    - **High Temperature Warning**
    - **Seatbelt Warning**
    - **High Beam Indicator**
    - **Turn Signal Indicators**
3. **User Input (Minimal):**
    - **Trip Reset Button (for odometer/trip meter)**
4. **Communication:**
    - **Receive vehicle data from other ECUs (Engine, BCM, ABS) via CAN (or simulated).**
    - **Send status (e.g., trip reset confirmation) via CAN (or simulated).**

II. Functional Blocks of the Dashboard Panel ECU:

1. **Microcontroller (MCU) Core:**
    - **Processor**
    - **Memory**
    - **Timers/Counters**
    - **ADC**
    - **GPIOs**
    - **Communication Peripherals (e.g., CAN)**

2. **Input Conditioning & Interface:**
    - **Sensor Data Inputs:**
        - **Vehicle Speed**
        - **Engine RPM**
        - **Fuel Level Sensor**
        - **Engine Temperature Sensor**
    - **Warning Light Status**
    - **High Beam/Turn Signal Inputs**
    - **Trip Reset Button**
3. **Output Drivers & Interfaces:**
    - **Display Drivers:**
        - **Segmented LCD/VFD Driver**
        - **Graphic LCD Controller**
        - **Stepper Motor Drivers**
    - **LED Drivers**
    - **Buzzer/Chime Driver**
    - **Backlighting Control**
4. **Power Management:**
    - **Voltage regulators (e.g., 5V, 3.3V).**
    - **Reverse polarity and transient protection.**
    - **Filtering for noise.**
5. **Communication Interface (e.g., CAN Transceiver):**
    - **CAN transceiver (e.g., NXP TJA1051) for physical layer.**
    - **CAN controller (integrated in MCU or external like MCP2515 for 8051).**

III. Software Logic (Firmware):

- **Initialization**
- **Input Acquisition**
- **Data Processing:**
    - **Scaling & Conversion**

    - **Filtering**
- **Display Update Logic:**
    - **Gauge Control**
    - **Numeric Display Update**
    - **Warning Light Logic**
- **User Interface Logic**
- **Diagnostic Logic**
- **Communication Stack (CAN)**
- **Watchdog Timer**

Keil-u-Microvision CODE

```
#include <reg51.h> // Standard 8051 register definitions
#include <stdio.h> // For sprintf()

// --- Pin Definitions for LCD ---
#define LCD_DATA_PORT P0      // P0 connected to LCD Data pins (D0-D7)
sbit LCD_RS = P2^0;           // Register Select
sbit LCD_RW = P2^1;           // Read/Write (connect to GND for write-only)
sbit LCD_EN = P2^2;           // Enable

// --- Pin Definitions for Simulated ADC ---
// Assuming ADC0804 connected to P3 as data port for reading
// And control pins for simplicity on P2. You might need more pins for multiple ADC inputs.
#define ADC_DATA_IN P3        // P3 connected to ADC Data Out pins
sbit ADC_RD_PIN = P2^4;       // ADC Read pin
sbit ADC_WR_PIN = P2^5;       // ADC Write/Start Conversion pin
sbit ADC_INTR_PIN = P2^7;     // ADC Interrupt/End of Conversion pin (active low)
```

```
// --- Pin Definitions for Warnings ---
sbit LOW_FUEL_LED   = P2^3; // Low Fuel Warning LED (Active High)
sbit CHECK_ENGINE_SW = P2^6; // Check Engine Light Input (Active Low, simulated)

// --- Global Variables ---
unsigned int ms_timer = 0;   // Millisecond counter for general timing
char lcd_buffer[17];        // Buffer for LCD display (16 chars + null terminator)

// --- Function Prototypes ---
void delay_ms(unsigned int ms);
void timer0_isr(void) interrupt 1;

// LCD Functions
void lcd_cmd(unsigned char cmd);
void lcd_data(unsigned char dat);
void lcd_init(void);
void lcd_string(char *s);
void lcd_set_cursor(unsigned char row, unsigned char col);

// ADC Functions (simplified for ADC0804 with single channel)
unsigned char read_adc(void);

// --- Main Program ---
void main() {
   // Initialize outputs
   LOW_FUEL_LED = 0;

   // Initialize LCD
   lcd_init();
```

```
    lcd_string("Dash Panel Init");
    lcd_set_cursor(1, 0);
    lcd_string("Loading...");
    delay_ms(1000); // Display loading message for a while

    // Configure Timer 0 for 1ms interrupt
    TMOD = 0x01; // Timer 0, Mode 1 (16-bit timer)
    TH0 = 0xFC;  // For 11.0592 MHz, 1ms reload value
    TL0 = 0x17;
    ET0 = 1;     // Enable Timer 0 interrupt
    EA = 1;      // Enable global interrupts
    TR0 = 1;     // Start Timer 0

    while(1) {
        // --- Read Simulated Sensor Data (via ADC) ---
        // For simplicity, using one ADC input for Speed, and then simulating another.
        // In a real setup, you'd likely use an ADC with multiple input channels or multiple ADC chips.
        unsigned char raw_speed = read_adc(); // Read "speed" from ADC input
        unsigned int vehicle_speed_kmh = (unsigned int)(raw_speed * 200 / 255); // Scale 0-255 to 0-200 km/h

        // Simulate Fuel Level (e.g., if ADC input was tied to a different pin for fuel)
        // For this example, let's just use raw_speed/2 for "fuel"
        unsigned char raw_fuel = raw_speed / 2;
        unsigned int fuel_percent = (unsigned int)(raw_fuel * 100 / 255); // Scale 0-255 to 0-100%

        // --- Warning Light Logic ---
        // Low Fuel Warning
```

```
        if (fuel_percent < 15) { // If fuel below 15%
            LOW_FUEL_LED = 1; // Turn on LED
        } else {
            LOW_FUEL_LED = 0; // Turn off LED
        }

        // Check Engine Warning (simulated via switch input)
        if (CHECK_ENGINE_SW == 0) { // If switch is pressed (active low)
            // In a real system, this would be a CAN message.
            // For 8051, let's just use the STATUS_LED on P2.2 for this specific warning.
            STATUS_LED = 1; // Turn on Check Engine LED (simulated)
        } else {
            STATUS_LED = 0;
        }

        // --- Display Updates ---
        // Display Speed on Line 0
        lcd_set_cursor(0, 0);
        sprintf(lcd_buffer, "Speed:%3u km/h", vehicle_speed_kmh);
        lcd_string(lcd_buffer);

        // Display Fuel on Line 1
        lcd_set_cursor(1, 0);
        sprintf(lcd_buffer, "Fuel: %3u%% ", fuel_percent);
        lcd_string(lcd_buffer);

        delay_ms(200); // Update display every 200ms
    }
}
```

```
// --- Function Implementations ---

// Simple blocking delay function
void delay_ms(unsigned int ms) {
   unsigned int i, j;
   for (i = 0; i < ms; i++) {
      for (j = 0; j < 1200; j++); // Adjust for 1ms @ 11.0592 MHz
   }
}

// Timer 0 Interrupt Service Routine (triggered every 1ms)
void timer0_isr(void) interrupt 1 {
   // Reload timer for next 1ms
   TH0 = 0xFC;
   TL0 = 0x17;

   ms_timer++; // Increment global millisecond counter
   // Could add other time-based tasks here (e.g., software debouncing)
}

// --- LCD Functions (HD44780 compatible) ---
void lcd_cmd(unsigned char cmd) {
   LCD_DATA_PORT = cmd; // Send command to data port
   LCD_RS = 0;         // RS=0 for command
   LCD_RW = 0;         // RW=0 for write
   LCD_EN = 1;         // Enable pulse
   delay_ms(1);        // Short delay
   LCD_EN = 0;         // Disable pulse
```

```
    if (cmd < 4) delay_ms(2); // Longer delay for clear display or return home
}

void lcd_data(unsigned char dat) {
    LCD_DATA_PORT = dat; // Send data to data port
    LCD_RS = 1;         // RS=1 for data
    LCD_RW = 0;         // RW=0 for write
    LCD_EN = 1;         // Enable pulse
    delay_ms(1);        // Short delay
    LCD_EN = 0;         // Disable pulse
    delay_ms(1);        // Short delay
}

void lcd_init(void) {
    delay_ms(100); // Power-on delay
    lcd_cmd(0x38); // 2 lines, 5x7 matrix
    lcd_cmd(0x0C); // Display ON, Cursor OFF, Blink OFF
    lcd_cmd(0x06); // Increment cursor (shift cursor to right)
    lcd_cmd(0x01); // Clear display screen
    delay_ms(2);   // Longer delay for clear display
}

void lcd_string(char *s) {
    while (*s) {
        lcd_data(*s++);
    }
}

void lcd_set_cursor(unsigned char row, unsigned char col) {
```

```
    unsigned char address;
    if (row == 0) {
        address = 0x80 + col; // First row
    } else {
        address = 0xC0 + col; // Second row
    }
    lcd_cmd(address);
}

// --- ADC Function (Simplified for ADC0804 single channel) ---
unsigned char read_adc(void) {
    unsigned char adc_value;

    // Start conversion
    ADC_RD_PIN = 1; // RD high initially
    ADC_WR_PIN = 0; // WR low to start conversion
    delay_ms(1);    // Small pulse width
    ADC_WR_PIN = 1; // WR high

    // Wait for end of conversion (INTR goes low)
    while (ADC_INTR_PIN != 0); // Loop until INTR goes low

    // Read data
    ADC_RD_PIN = 0; // RD low to enable output
    delay_ms(1);    // Small delay for data to become stable
    adc_value = ADC_DATA_IN; // Read data from P3
    ADC_RD_PIN = 1; // RD high to disable output
    return adc_value;
}
```