Session-8 - ECU Design for Wiper Control

A Wiper Control ECU is responsible for managing the windshield wipers (and often the washer system) based on user input, vehicle speed, and sometimes environmental conditions (rain sensor).

I. High-Level Requirements

II. Functional Blocks of the Wiper Control ECU

III. Software Logic (Firmware)

IV. Mechanical and Environmental Considerations

V. Development Tools

→Design keil microvision of 8051 code for wiper control

```
#include <reg51.h> // Include standard 8051 register definitions

// --- Pin Definitions ---

// Wiper Stalk Inputs (Active Low)

sbit OFF_SW      = P1^0; // Wiper Off switch

sbit INTERMITT_SW = P1^1; // Intermittent mode switch

sbit LOW_SW      = P1^2; // Low speed switch

sbit HIGH_SW     = P1^3; // High speed switch

sbit WASHER_SW    = P1^4; // Washer switch

// Wiper Motor Relay Outputs (Active High)

sbit LOW_SPEED_RELAY  = P2^0; // Relay for low speed

sbit HIGH_SPEED_RELAY = P2^1; // Relay for high speed

// Washer Pump Relay Output (Active High)

sbit WASHER_PUMP_RELAY = P2^3;

// Wiper Park Position Sensor (Active Low when in Park)

sbit PARK_SENSOR = P3^2; // This pin is INT0 in 8051. Useful for interrupt based park detection.

// --- Global Variables ---

unsigned int intermittent_delay_counter = 0;

const unsigned int INTERMITTENT_PERIOD_MS = 3000; // 3 seconds for intermittent wipe

unsigned char wipe_count_after_wash = 0;

#define WIPE_COUNT_AFTER_WASH_MAX 3 // Number of wipes after washer spray
```

```
// --- Function Prototypes ---
void delay_ms(unsigned int ms);
void stop_wipers(void);
void start_low_speed(void);
void start_high_speed(void);
void perform_single_wipe(void);
void timer0_isr(void) interrupt 1; // Timer 0 Interrupt Service Routine

// --- Main Program ---
void main() {
  // Initialize all outputs to OFF state
  stop_wipers();
  WASHER_PUMP_RELAY = 0;
  // Configure Timer 0 for 1ms interrupt (Mode 1, 16-bit timer)
  // For 11.0592 MHz crystal, machine cycle is 1.085 us.
  // To get 1ms delay: (65536 - (1ms / 1.085us)) = 65536 - 921 = 64615 (DEC) = FC17 (HEX)
  TMOD = 0x01; // Timer 0, Mode 1 (16-bit timer)
  TH0 = 0xFC;  // Load higher byte
  TL0 = 0x17;  // Load lower byte
  ET0 = 1;     // Enable Timer 0 interrupt
  EA = 1;      // Enable global interrupts
  TR0 = 1;     // Start Timer 0
  while(1) {
    // --- Input Polling and State Management ---
    // Prioritize OFF
    if (OFF_SW == 0) { // If OFF switch is pressed (active low)
      stop_wipers();
      intermittent_delay_counter = 0; // Reset counter
      wipe_count_after_wash = 0; // Reset wash wipe counter
      WASHER_PUMP_RELAY = 0; // Ensure washer is off
      while(OFF_SW == 0); // Debounce / wait for release
```

```
    delay_ms(50); // Small debounce delay
}
// Washer Mode
else if (WASHER_SW == 0) { // If WASHER switch is pressed
    stop_wipers(); // Stop wipers first
    WASHER_PUMP_RELAY = 1; // Turn on washer pump
    delay_ms(1000); // Keep pump on for 1 second
    WASHER_PUMP_RELAY = 0; // Turn off pump
    perform_single_wipe(); // Perform one wipe immediately
    wipe_count_after_wash = 1; // Start post-wash wipe counter
    intermittent_delay_counter = 0; // Reset intermittent timer for subsequent wipes
    while(WASHER_SW == 0); // Debounce / wait for release
    delay_ms(50); // Small debounce delay
}
// High Speed Mode
else if (HIGH_SW == 0) {
    start_high_speed();
    intermittent_delay_counter = 0;
    wipe_count_after_wash = 0;
}
// Low Speed Mode
else if (LOW_SW == 0) {
    start_low_speed();
    intermittent_delay_counter = 0;
    wipe_count_after_wash = 0;
}
// Intermittent Mode
else if (INTERMITT_SW == 0) {
    // Intermittent logic handled by Timer 0 ISR
    // Wipers are off by default in intermittent until the period elapses
    stop_wipers();
```

```
            wipe_count_after_wash = 0; // Clear any pending wash wipes
        }
        // Default to Off if no other mode is selected (or after a mode finishes)
        else {
            stop_wipers();
            intermittent_delay_counter = 0;
            wipe_count_after_wash = 0;
        }
        // Small delay to prevent excessive polling
        delay_ms(10);
    }
}
// --- Function Implementations ---
// Simple delay function (blocking)
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for (i = 0; i < ms; i++) {
        for (j = 0; j < 1200; j++); // Adjust this loop for actual 1ms on your specific 8051 @ 11.0592 MHz
                                    // A more accurate delay would use timers.
    }
}
// Stops both wiper motor relays
void stop_wipers(void) {
    // If wipers are not at park, continue until they reach park
    // This is a crucial part of actual wiper control
    if (LOW_SPEED_RELAY == 1 || HIGH_SPEED_RELAY == 1) { // If motor is active
        start_low_speed(); // Force to low speed to reach park
        while(PARK_SENSOR != 0); // Wait until park sensor is active (low)
    }
    LOW_SPEED_RELAY = 0;
```

```
    HIGH_SPEED_RELAY = 0;
}
// Starts low speed wipers
void start_low_speed(void) {
    HIGH_SPEED_RELAY = 0; // Ensure high speed is off
    LOW_SPEED_RELAY = 1;
}
// Starts high speed wipers
void start_high_speed(void) {
    LOW_SPEED_RELAY = 0; // Ensure low speed is off
    HIGH_SPEED_RELAY = 1;
}
// Performs a single wipe cycle
void perform_single_wipe(void) {
    start_low_speed(); // Start wipers at low speed
    // Wait for the wipers to leave the park position and return
    // This assumes the park sensor goes high when leaving park and low when returning
    while (PARK_SENSOR == 0); // Wait for wipers to leave park (sensor goes high)
    while (PARK_SENSOR != 0); // Wait for wipers to return to park (sensor goes low)
    stop_wipers(); // Turn off motor
}
// Timer 0 Interrupt Service Routine (triggered every 1ms)
void timer0_isr(void) interrupt 1 {
    // Reload timer for next 1ms
    TH0 = 0xFC;
    TL0 = 0x17;
    // --- Intermittent Wipe Logic ---
    if (INTERMITT_SW == 0) { // Only active if intermittent mode is selected
        intermittent_delay_counter++;
        if (intermittent_delay_counter >= INTERMITTENT_PERIOD_MS) {
            perform_single_wipe();
```

```
            intermittent_delay_counter = 0; // Reset counter after wipe
        }
    } else {
        intermittent_delay_counter = 0; // Reset if not in intermittent mode
    }
    // --- Post-Wash Wipe Logic ---
    if (wipe_count_after_wash > 0) {
        // After washer, wait for a short period before next wipe
        // This simple example just triggers wipes without much delay
        // A more robust implementation would use a separate timer for post-wash delays
        if (intermittent_delay_counter % 500 == 0 && intermittent_delay_counter != 0) { // Example:
wipe every 0.5 sec after wash
            perform_single_wipe();
            wipe_count_after_wash--;
        }
    }
}
// Add an external interrupt service routine for the park sensor if desired for robustness
/*
void park_sensor_isr(void) interrupt 0 // External Interrupt 0 (INT0)
{
    // This ISR would be useful if you need immediate reaction to the park sensor
    // e.g., to stop the motor very precisely when it hits park.
    // In this example, we poll it, but an ISR is more robust for critical timing.
}
*/
```