ECU Design for Mirror Control

A mirror control ECU is a small, dedicated microcontroller-based unit responsible for interpreting the driver's commands and precisely moving the side-view mirrors. In modern vehicles, this ECU is often a node on a larger vehicle network like the **CAN (Controller Area Network) bus**, receiving commands from a central Body Control Module (BCM).

For this example, we will simplify it to a standalone system where the switches are directly connected to the ECU's input pins.

---

1. Conceptual Hardware Design ⚙️

To build this system, we need a few key components:

- **Microcontroller (The ECU's Brain):** A simple 8-bit or 32-bit microcontroller (like an ATmega328P or an ARM Cortex-M0) is sufficient. It will read inputs and send output signals.
- **Driver Inputs:**
    - **Mirror Select Switch:** A single-pole, double-throw (SPDT) switch to select either the **LEFT** or **RIGHT** mirror.
    - **Joystick/4-Way Switch:** Four simple push buttons (or a 4-way hat switch) for directional control: **UP**, **DOWN**, **LEFT**, and **RIGHT**.
- **Motor Driver (H-Bridge):** A microcontroller's GPIO pins cannot provide enough current to drive a motor. More importantly, we need to reverse the motor's direction. An **H-Bridge** IC (like the L293D) is perfect for this. It allows us to control both the speed and direction of two separate DC motors. Each mirror contains two motors: one for vertical (tilt) movement and one for horizontal (pan) movement.
- **DC Motors:** Two small DC motors are embedded within each mirror assembly.

Signal Flow: Driver uses Joystick → Microcontroller reads switch states → Microcontroller sends signals to H-Bridge → H-Bridge drives the correct motor in the correct direction.

 CAN Bus Integration: Instead of direct pin reads, the ECU would listen for CAN messages from the Body Control Module. For example, it might receive a CAN frame with ID 0x250 where the data bytes indicate which mirror and direction to move.

 Position Memory: For memory seats/mirrors, potentiometers would be attached to the mirror's pivot points. The ECU would read the analog voltage from these sensors to know the mirror's exact position, allowing it to store and recall specific locations for different drivers.

 Switch Debouncing: A software algorithm or hardware filter would be used to handle the "bouncing" of physical switch contacts to ensure one button press results in one action.

 Diagnostics and Safety: The ECU would monitor the current drawn by the motors. If a motor stalls (e.g., it's at its limit or blocked), the current will spike. The ECU would detect this over-current condition, cut power to the motor to prevent damage, and log a Diagnostic Trouble Code (DTC) that a technician can read.

 Power Management: The ECU must enter a low-power "sleep" mode when the vehicle is off, waking up only when required (e.g., when the ignition is turned on or a CAN message is received).

```
#include <stdint.h> // For standard integer types like uint8_t
#include <stdbool.h> // For boolean type (true/false)
// --- INPUT PINS from Driver Controls ---
// Assuming active LOW (pressed = 0)
#define MIRROR_SELECT_PIN   0  // Input: 0 for LEFT, 1 for RIGHT
#define JOYSTICK_UP_PIN     1  // Input: 0 when pressed
#define JOYSTICK_DOWN_PIN   2  // Input: 0 when pressed
#define JOYSTICK_LEFT_PIN   3  // Input: 0 when pressed
#define JOYSTICK_RIGHT_PIN  4  // Input: 0 when pressed
// --- OUTPUT PINS to H-Bridge Motor Driver ---
// Left Mirror Motors
#define LEFT_H_MOTOR_IN1    8  // Horizontal motor control 1
#define LEFT_H_MOTOR_IN2    9  // Horizontal motor control 2
#define LEFT_V_MOTOR_IN1    10 // Vertical motor control 1
#define LEFT_V_MOTOR_IN2    11 // Vertical motor control 2
// Right Mirror Motors
#define RIGHT_H_MOTOR_IN1   12 // Horizontal motor control 1
#define RIGHT_H_MOTOR_IN2   13 // Horizontal motor control 2
#define RIGHT_V_MOTOR_IN1   14 // Vertical motor control 1
#define RIGHT_V_MOTOR_IN2   15 // Vertical motor control 2
typedef enum {
    LEFT_MIRROR,
    RIGHT_MIRROR
} MirrorSelection_t;
typedef enum {
    DIR_UP,
    DIR_DOWN,
    DIR_LEFT,
    DIR_RIGHT,
    DIR_NONE
} JoystickDirection_t;
typedef enum {
    FORWARD,
```

```
    BACKWARD,
    STOP
} MotorState_t;
// Initializes GPIO pins as either input or output
void init_gpio(void) {
    // In a real MCU, you would configure data direction registers (e.g., DDRx in AVR)
    // For pins 0-4 (Inputs)
    // For pins 8-15 (Outputs)
    printf("GPIO Initialized.\n");
}
// Reads the logic level of a given pin
bool read_pin(uint8_t pin) {
    // This is a placeholder for reading a GPIO register (e.g., PINx in AVR)
    // We'll simulate this in the main loop for demonstration.
    return true; // Return true (HIGH) by default
}
// Writes a logic level to a given pin
void write_pin(uint8_t pin, bool level) {
    // This is a placeholder for writing to a GPIO register (e.g., PORTx in AVR)
    printf("Pin %d set to %s\n", pin, level ? "HIGH" : "LOW");
}
/**
 * @brief Controls a single motor via its two H-Bridge input pins.
 * @param state The desired state (FORWARD, BACKWARD, STOP).
 * @param pin1 H-Bridge input 1.
 * @param pin2 H-Bridge input 2.
 */
void control_motor(MotorState_t state, uint8_t pin1, uint8_t pin2) {
    switch(state) {
        case FORWARD:
            write_pin(pin1, true);  // HIGH
            write_pin(pin2, false); // LOW
            break;
```

```
        case BACKWARD:
            write_pin(pin1, false); // LOW
            write_pin(pin2, true);  // HIGH
            break;
        case STOP:
        default:
            write_pin(pin1, false); // LOW
            write_pin(pin2, false); // LOW (Brakes the motor)
            break;
    }
}
int main(void) {
    // 1. Initialization
    init_gpio();

    // 2. Main Superloop
    while(1) {
        // --- Read Inputs ---
        MirrorSelection_t selected_mirror = read_pin(MIRROR_SELECT_PIN) ? RIGHT_MIRROR : LEFT_MIRROR;
        JoystickDirection_t joy_direction = DIR_NONE;

        if (!read_pin(JOYSTICK_UP_PIN))    joy_direction = DIR_UP;
        if (!read_pin(JOYSTICK_DOWN_PIN))  joy_direction = DIR_DOWN;
        if (!read_pin(JOYSTICK_LEFT_PIN))  joy_direction = DIR_LEFT;
        if (!read_pin(JOYSTICK_RIGHT_PIN)) joy_direction = DIR_RIGHT;

        // --- Stop all motors by default ---
        control_motor(STOP, LEFT_H_MOTOR_IN1, LEFT_H_MOTOR_IN2);
        control_motor(STOP, LEFT_V_MOTOR_IN1, LEFT_V_MOTOR_IN2);
        control_motor(STOP, RIGHT_H_MOTOR_IN1, RIGHT_H_MOTOR_IN2);
        control_motor(STOP, RIGHT_V_MOTOR_IN1, RIGHT_V_MOTOR_IN2);

        // --- Process Logic ---
```

```
      if (selected_mirror == LEFT_MIRROR) {

         if (joy_direction == DIR_UP)    control_motor(FORWARD, LEFT_V_MOTOR_IN1, LEFT_V_MOTOR_IN2);

         if (joy_direction == DIR_DOWN)  control_motor(BACKWARD, LEFT_V_MOTOR_IN1,
LEFT_V_MOTOR_IN2);

         if (joy_direction == DIR_LEFT)  control_motor(FORWARD, LEFT_H_MOTOR_IN1, LEFT_H_MOTOR_IN2);

         if (joy_direction == DIR_RIGHT) control_motor(BACKWARD, LEFT_H_MOTOR_IN1, LEFT_H_MOTOR_IN2);

      }

      else { // RIGHT_MIRROR is selected

         if (joy_direction == DIR_UP)    control_motor(FORWARD, RIGHT_V_MOTOR_IN1, RIGHT_V_MOTOR_IN2);

         if (joy_direction == DIR_DOWN)  control_motor(BACKWARD, RIGHT_V_MOTOR_IN1,
RIGHT_V_MOTOR_IN2);

         if (joy_direction == DIR_LEFT)  control_motor(FORWARD, RIGHT_H_MOTOR_IN1,
RIGHT_H_MOTOR_IN2);

         if (joy_direction == DIR_RIGHT) control_motor(BACKWARD, RIGHT_H_MOTOR_IN1,
RIGHT_H_MOTOR_IN2);

      }


      // Add a small delay in a real system to prevent bouncing and reduce CPU load

      // delay_ms(50);

   }


   return 0;

}
```