Session 16 : ECU design for door locking system

ECU Design for Door Locking System (Simplified)

I. High-Level Requirements (Simplified for 8051 Example):

1. **Central Lock/Unlock: Activate all (or a single, in our case) door locks simultaneously.**
2. **User Input: An interior button for Lock/Unlock.**
3. **Lock/Unlock Status Feedback: Read the current state of the lock.**
4. **Actuator Control: Drive a motor/solenoid for lock/unlock action.**
5. **Basic Timeout: Prevent motor burnout if the lock jams.**

II. Hardware Assumptions for 8051 Code (Keil uVision):

- **Microcontroller: AT89C51 / AT89S52 (or compatible)**
- **Crystal Frequency: 11.0592 MHz (for accurate timing)**
- **Inputs (Active Low Switches):**
  - **P1.0: Central LOCK Button**
  - **P1.1: Central UNLOCK Button**
  - **P1.2: Lock Position Feedback (0=Locked, 1=Not Locked - e.g., a microswitch at the locked position)**
  - **P1.3: Unlock Position Feedback (0=Unlocked, 1=Not Unlocked - e.g., a microswitch at the unlocked position)**
- **Outputs (Active High - for H-bridge control or relays):**
  - **P2.0: Motor_A (one side of an H-bridge or a relay coil for lock)**
  - **P2.1: Motor_B (other side of an H-bridge or a relay coil for unlock)**
  - **P2.2: Status LED (1=Locked, 0=Unlocked)**
  - **P2.3: Error LED (1=Error, e.g., motor timeout)**

Keil-U-Microvision CODE :

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


// --- Pin Definitions ---
// Inputs (Active Low Switches)
sbit LOCK_BTN_IN     = P1^0; // Central LOCK button input
sbit UNLOCK_BTN_IN   = P1^1; // Central UNLOCK button input
sbit LOCK_FB_IN      = P1^2; // Lock Position Feedback (0=Locked)
sbit UNLOCK_FB_IN    = P1^3; // Unlock Position Feedback (0=Unlocked)

// Outputs (Active High for Motor Drivers / Relays)
sbit MOTOR_A         = P2^0; // Control line A for H-bridge / Lock Relay
sbit MOTOR_B         = P2^1; // Control line B for H-bridge / Unlock Relay
sbit STATUS_LED      = P2^2; // Status LED (ON for Locked, OFF for Unlocked)
sbit ERROR_LED       = P2^3; // Error LED (ON for motor timeout/fault)

// --- Global Variables ---
unsigned int ms_timer = 0;      // Millisecond counter for general timing
unsigned char current_lock_state = 0; // 0: Unlocked, 1: Locked, 2: Locking, 3: Unlocking, 4: Error
#define STATE_UNLOCKED  0
#define STATE_LOCKED    1
#define STATE_LOCKING   2
#define STATE_UNLOCKING 3
#define STATE_ERROR     4

#define MOTOR_TIMEOUT_MS  1000  // Max time for motor to complete action (1 second)

// --- Function Prototypes ---
void delay_ms(unsigned int ms);
void timer0_isr(void) interrupt 1;
void set_lock_system_state(unsigned char state);
void activate_lock_motor(unsigned char action); // 0=stop, 1=lock, 2=unlock

// --- Main Program ---
void main() {
```

```
// Initialize outputs to safe/off state
MOTOR_A = 0;
MOTOR_B = 0;
STATUS_LED = 0; // Start as Unlocked visually
ERROR_LED = 0;

// Configure Timer 0 for 1ms interrupt (Mode 1, 16-bit timer)
// For 11.0592 MHz crystal, reload values for 1ms are 0xFC17
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

// Determine initial lock state from feedback sensors
if (LOCK_FB_IN == 0 && UNLOCK_FB_IN == 1) { // Assuming LOCK_FB_IN is low when locked
   current_lock_state = STATE_LOCKED;
   STATUS_LED = 1; // Visual feedback
} else {
   current_lock_state = STATE_UNLOCKED;
   STATUS_LED = 0; // Visual feedback
}

while(1) {
   // --- Input Polling and Command Processing ---
   // Basic debounce: Check twice with a small delay. In a real system, use more robust debouncing.
   if (LOCK_BTN_IN == 0) { // If LOCK button is pressed (active low)
      delay_ms(20); // Debounce
      if (LOCK_BTN_IN == 0 && current_lock_state != STATE_LOCKED && current_lock_state != STATE_LOCKING) {
         set_lock_system_state(STATE_LOCKING);
      }
      while(LOCK_BTN_IN == 0); // Wait for button release
   }
```

```
    if (UNLOCK_BTN_IN == 0) { // If UNLOCK button is pressed (active low)
      delay_ms(20); // Debounce
      if (UNLOCK_BTN_IN == 0 && current_lock_state != STATE_UNLOCKED && current_lock_state != STATE_UNLOCKING) {
        set_lock_system_state(STATE_UNLOCKING);
      }
      while(UNLOCK_BTN_IN == 0); // Wait for button release
    }

    // --- Door Lock System State Machine ---
    switch (current_lock_state) {
      case STATE_LOCKED:
        STATUS_LED = 1; // Solid LED for Locked state
        ERROR_LED = 0;  // No error
        activate_lock_motor(0); // Ensure motor is off
        // Check if physical lock state matches expected locked state
        if (LOCK_FB_IN != 0 || UNLOCK_FB_IN != 1) {
          // Mismatch, potential issue, could transition to error or re-attempt lock
          // For simplicity, we just stay in locked but a real system would manage this.
        }
        break;

      case STATE_UNLOCKED:
        STATUS_LED = 0; // LED off for Unlocked state
        ERROR_LED = 0;  // No error
        activate_lock_motor(0); // Ensure motor is off
        // Check if physical lock state matches expected unlocked state
        if (LOCK_FB_IN != 1 || UNLOCK_FB_IN != 0) {
          // Mismatch, potential issue
        }
        break;

      case STATE_LOCKING:
        activate_lock_motor(1); // Turn motor for LOCK action
        ERROR_LED = 0;        // Clear error LED during operation
```

```
    // Check for completion of lock action
    if (LOCK_FB_IN == 0) { // If lock feedback indicates locked position
        set_lock_system_state(STATE_LOCKED); // Transition to LOCKED state
    }
    // Check for motor timeout (fault detection)
    else if (ms_timer >= MOTOR_TIMEOUT_MS) {
        activate_lock_motor(0);        // Stop motor
        set_lock_system_state(STATE_ERROR); // Transition to ERROR state
        // In a real system: log Diagnostic Trouble Code (DTC), potentially try again
    }
    break;

case STATE_UNLOCKING:
    activate_lock_motor(2); // Turn motor for UNLOCK action
    ERROR_LED = 0;         // Clear error LED during operation

    // Check for completion of unlock action
    if (UNLOCK_FB_IN == 0) { // If unlock feedback indicates unlocked position
        set_lock_system_state(STATE_UNLOCKED); // Transition to UNLOCKED state
    }
    // Check for motor timeout (fault detection)
    else if (ms_timer >= MOTOR_TIMEOUT_MS) {
        activate_lock_motor(0);        // Stop motor
        set_lock_system_state(STATE_ERROR); // Transition to ERROR state
        // In a real system: log DTC, trigger warning
    }
    break;

case STATE_ERROR:
    activate_lock_motor(0); // Ensure motor is off
    STATUS_LED = (ms_timer % 500 < 250) ? 1 : 0; // Flash STATUS_LED for error indication
    ERROR_LED = 1; // Solid ERROR_LED

    // Logic to clear error state (e.g., specific reset input, power cycle)
```

```
            // For this example, stays in error until another valid command is given.
            break;

        default:
            set_lock_system_state(STATE_ERROR); // Unknown state, go to error
            break;
    }

    delay_ms(50); // Small delay to prevent excessive polling loop
  }
}

// --- Function Implementations ---

// Simple blocking delay function (for debouncing and short waits)
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. Use timers for precise long delays.
  }
}

// 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
}

// Sets the current lock system state and resets the motor timeout timer
void set_lock_system_state(unsigned char state) {
  current_lock_state = state;
  ms_timer = 0; // Reset timeout timer on state change
```

```
}

// Controls the motor direction for lock/unlock
// action: 0=stop, 1=lock (MOTOR_A high), 2=unlock (MOTOR_B high)
void activate_lock_motor(unsigned char action) {
  if (action == 1) { // Command to LOCK
    MOTOR_B = 0; // Ensure opposite line is off
    MOTOR_A = 1;
  } else if (action == 2) { // Command to UNLOCK
    MOTOR_A = 0; // Ensure opposite line is off
    MOTOR_B = 1;
  } else { // Command to STOP
    MOTOR_A = 0;
    MOTOR_B = 0;
  }
}
```