Session 12 - ECU design for door locking system

I. High-Level Requirements:

1. Central Locking/Unlocking
2. Individual Door Lock/Unlock
3. Door Status Monitoring
4. Lock/Unlock Status Feedback
5. User Inputs
6. Automatic Relocking (Optional)
7. Automatic Locking (Optional)
8. Anti-Pinch (Optional for power windows)
9. Diagnostic Capabilities
10. Communication

II. Functional Blocks of the Door Locking ECU:

1. Microcontroller (MCU) Core
2. Input Conditioning & Interface
3. Output Drivers & Interfaces
4. Power Management
5. Communication Interface (e.g., CAN Transceiver)

III. Software Logic (Firmware):

- Initialization
- Input Monitoring
- State Machine for Each Door Lock
- Lock/Unlock Control Logic
- Central Locking Coordination
- Automatic Locking/Relocking Logic
- Diagnostic Logic
- Communication Handler

- Watchdog Timer

→Keil-u-Microvision CODE:-

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


// --- Pin Definitions ---
// Inputs (Active Low)
sbit LOCK_BTN        = P1^0; // Interior Lock Button
sbit UNLOCK_BTN      = P1^1; // Interior Unlock Button
sbit DOOR_OPEN_SW    = P1^2; // Door Open/Closed Switch (0=Open, 1=Closed)
sbit LOCK_FB         = P1^3; // Lock Position Feedback (0=Locked, 1=Unlocked)
sbit UNLOCK_FB       = P1^4; // Unlock Position Feedback (0=Unlocked, 1=Locked)
sbit SPEED_SENSE     = P1^5; // Simulated Speed Sensor (0=High, 1=Low)

// Outputs (Active High for Motor Drivers / Relays)
sbit MOTOR_FWD       = P2^0; // Controls motor for LOCK action
sbit MOTOR_REV       = P2^1; // Controls motor for UNLOCK action
sbit STATUS_LED      = P2^2; // Simple LED for lock status (1=Locked, 0=Unlocked)

// --- Global Variables ---
unsigned int ms_timer = 0;      // Millisecond counter
unsigned int auto_relock_timer = 0;
unsigned char current_lock_state = 0; // 0: Unlocked, 1: Locked, 2: Locking, 3: Unlocking
#define UNLOCKED 0
#define LOCKED   1
#define LOCKING  2
#define UNLOCKING 3

#define MOTOR_TIMEOUT_MS  500  // Max time for motor to complete action (0.5 seconds)
#define AUTO_RELOCK_DELAY_MS 10000 // 10 seconds for auto-relock
```

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

// --- Main Program ---
void main() {
  // Initialize outputs to safe/off state
  MOTOR_FWD = 0;
  MOTOR_REV = 0;
  STATUS_LED = 0; // Assume unlocked initially

  // 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

  // Initial state: Read actual lock feedback to set current_lock_state
  if (LOCK_FB == 0 && UNLOCK_FB == 1) { // Assuming LOCK_FB is low when locked
    current_lock_state = LOCKED;
    STATUS_LED = 1;
  } else {
    current_lock_state = UNLOCKED;
    STATUS_LED = 0;
```

```
    }

    while(1) {
        // --- Input Debouncing & Command Processing ---
        // Simple debouncing: read twice with a short delay
        if (LOCK_BTN == 0) { // Lock button pressed
            delay_ms(20); // Debounce
            if (LOCK_BTN == 0 && current_lock_state != LOCKED && current_lock_state != LOCKING) {
                set_lock_state(LOCKING);
            }
            while(LOCK_BTN == 0); // Wait for release
        }

        if (UNLOCK_BTN == 0) { // Unlock button pressed
            delay_ms(20); // Debounce
            if (UNLOCK_BTN == 0 && current_lock_state != UNLOCKED && current_lock_state != UNLOCKING) {
                set_lock_state(UNLOCKING);
                auto_relock_timer = 0; // Reset auto-relock on intentional unlock
            }
            while(UNLOCK_BTN == 0); // Wait for release
        }

        // --- State Machine for Lock Actuator ---
        switch (current_lock_state) {
            case LOCKED:
                STATUS_LED = 1; // Indicate locked
                activate_lock_motor(0); // Ensure motor is off
```

```
            // Check for auto-lock on speed (simplified)
            if (SPEED_SENSE == 0 && DOOR_OPEN_SW == 1) { // If high speed & door closed
                // Already locked, do nothing
            }
            break;

        case UNLOCKED:
            STATUS_LED = 0; // Indicate unlocked
            activate_lock_motor(0); // Ensure motor is off

            // Auto Relock Logic: If unlocked and no door opened within timeout
            if (auto_relock_timer > 0 && auto_relock_timer >= AUTO_RELOCK_DELAY_MS) {
                if (DOOR_OPEN_SW == 1) { // If door is closed (meaning no one opened it)
                    set_lock_state(LOCKING); // Trigger auto-relock
                    auto_relock_timer = 0; // Reset timer
                } else {
                    auto_relock_timer = 0; // Reset if door is open (cancel auto-relock)
                }
            }
            // Auto-lock on speed (simplified) - only lock if currently unlocked
            else if (SPEED_SENSE == 0 && DOOR_OPEN_SW == 1) { // High speed & door
closed
                set_lock_state(LOCKING); // Trigger auto-lock
            }
            break;

        case LOCKING:
            activate_lock_motor(1); // Turn motor to LOCK direction
            if (LOCK_FB == 0) { // Check if lock feedback indicates locked position
                set_lock_state(LOCKED); // Transition to LOCKED state
```

```
            }
            // Motor Timeout (fault detection)
            if (ms_timer >= MOTOR_TIMEOUT_MS) {
                // Motor stalled or blocked, report fault (simplified: just stop)
                activate_lock_motor(0);
                set_lock_state(UNLOCKED); // Revert to unlocked state or error state
                // In a real system: log DTC, trigger warning
            }
            break;

        case UNLOCKING:
            activate_lock_motor(2); // Turn motor to UNLOCK direction
            if (UNLOCK_FB == 0) { // Check if unlock feedback indicates unlocked position
                set_lock_state(UNLOCKED); // Transition to UNLOCKED state
                auto_relock_timer = 1; // Start auto-relock timer
            }
            // Motor Timeout (fault detection)
            if (ms_timer >= MOTOR_TIMEOUT_MS) {
                // Motor stalled or blocked
                activate_lock_motor(0);
                set_lock_state(LOCKED); // Revert to locked state or error state
                // In a real system: log DTC, trigger warning
            }
            break;
    }

    delay_ms(10); // Small delay for main loop iteration
  }
}
```

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

// Simple delay function
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for (i = 0; i < ms; i++) {
        for (j = 0; j < 1200; j++); // Approx. 1ms for 11.0592MHz. Fine tune if needed.
    }
}

// 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 for motor timeouts
    if (auto_relock_timer > 0) { // Only increment if auto-relock is active
        auto_relock_timer++;
    }
}

// Sets the current lock state and resets the motor timeout timer
void set_lock_state(unsigned char state) {
    current_lock_state = state;
    ms_timer = 0; // Reset timeout timer whenever state changes
}
// Controls the motor direction
```

```
void activate_lock_motor(unsigned char direction) {
    if (direction == 1) { // Lock
        MOTOR_REV = 0; // Ensure opposite direction is off
        MOTOR_FWD = 1;
    } else if (direction == 2) { // Unlock
        MOTOR_FWD = 0; // Ensure opposite direction is off
        MOTOR_REV = 1;
    } else { // Stop
        MOTOR_FWD = 0;
        MOTOR_REV = 0;
    }
}
```