Use Case ESP32 Sensors (15 Channels) - Hệ Thống Quản Lý Bãi Đỗ Xe Thông Minh

Mô Tả Tổng Quát

ESP32 là cảm biến trung tâm quản lý **15 cảm biến siêu âm** (HC-SR04) thông qua **74HC595 Shift Register** + **CD74HC4067 MUX (16:1)** để phát hiện xe đỗ. ESP32 kết nối WiFi AP từ UNO R4 để gửi/nhận dữ liệu với Backend qua API.

Kiến trúc phần cứng:

- 15 cảm biến HC-SR04 (địa chỉ 1-15)
- 1 chân TRIG/ECHO chung (PIN_TRIG=25, PIN_ECHO=26)
- CD74HC4067 MUX 16:1 để chọn channel (S0-S3: GPIO 12,13,14,27)
- 74HC595 Shift Register để điều khiển relay/MOSFET (bit-bang: DS=23, SH=18, ST=5)
- WebServer port 80 cung cấp 2 endpoint: GET /data, POST /detect

---

Use Case 1: Kết Nối WiFi AP (WiFi Access Point)

Trường	Nội Dung
Tên Use Case	Kết Nối WiFi AP - ESP32 Kết Nối UNO R4
Mô Tả	ESP32 kết nối vào WiFi AP "UNO-R4-AP" phát từ UNO R4 (IP: 192.168.4.2), cho phép giao tiếp với Backend ở 192.168.4.3:5000.
Tác Nhân	ESP32 (Hardware), UNO R4 (WiFi AP), Backend Server
Mô Tả Chi Tiết	ESP32 WiFi.begin("UNO-R4-AP", "12345678"). Cấu hình IP tĩnh: 192.168.4.5. Kiểm tra kết nối mỗi 30s. Khởi động WebServer port 80.
Điều Kiện Tiên Quyết	UNO R4 phát AP "UNO-R4-AP"; ESP32 firmware config SSID/password; Backend chạy 192.168.4.3:5000
Luồng Chính	1. setup() khởi động 2. Khởi tạo Serial (115200) 3. Setup pin 74HC595 (DS, SH, ST), MUX pins (S0-S1-S2-S3), TRIG/ECHO 4. WiFi.config(IP tĩnh) 5. WiFi.begin(ssid, password) 6. Loop retry kết nối (max 20 lần) 7. Nếu WiFi ok, khởi động WebServer server.begin() 8. server.on("/data", handleGetData) 9. server.on("/detect", handleDetect) 10. Sẵn sàng nhận HTTP request
Luồng Thay Thế	A1: Local WiFi - Kết nối router thay vì AP

Trường	Nội Dung
Luồng Ngoài Lề	E1: WiFi init fail → log error, retry sau 30s E2: WebServer fail → log nhưng tiếp tục
Điều Kiện Sau	WiFi kết nối thành công; WebServer sẵn sàng port 80; Backend có thể giao tiếp

Use Case 2: Quét 15 Cảm Biến (15 Channels via MUX + Shift Register)

Trường	Nội Dung
Tên Use Case	Quét 15 Cảm Biến Siêu Âm - HC-SR04 via CD74HC4067 MUX
Mô Tả	ESP32 liên tục quét 15 cảm biến HC-SR04 thông qua CD74HC4067 MUX 16:1. Mỗi cảm biến được chọn bằng tín hiệu MUX (S0-S3), rồi đọc khoảng cách và lưu vào mảng currentDistances[15].
Tác Nhân	ESP32 (Hardware), HC-SR04 Sensors (15x), CD74HC4067 MUX, 74HC595 Shift Register
Mô Tả Chi Tiết	15 cảm biến HC-SR04 chia sẻ 1 chân TRIG/ECHO (PIN_TRIG=25, PIN_ECHO=26). CD74HC4067 MUX 16:1 (điều khiển bằng 4 chân S0-S3) chọn 1 trong 16 channel ECHO. 74HC595 Shift Register (bit-bang) điều khiển relay/MOSFET để cấp nguồn cho từng cảm biến. docTatCaCamBien() lặp qua 15 channel, set MUX, phát TRIG pulse, đợi ECHO, tính distance.
Điều Kiện Tiên Quyết	15 cảm biến HC-SR04 kết nối đúng; MUX 16:1 config; 74HC595 shift register setup; ESP32 khởi động

Trường	Nội Dung
Luồng Chính	1. docTatCaCamBien() gọi từ handleDetect() hoặc periodically 2. Lặp qua 15 cảm biến (ch = 0 to 14) 3. Set MUX channel bằng selectMuxChannel(ch): - digitalWrite(S0, ch & 1) - digitalWrite(S1, (ch>>1) & 1) - digitalWrite(S2, (ch>>2) & 1) - digitalWrite(S3, (ch>>3) & 1) 4. Bật relay/MOSFET cho cảm biến hiện tại bằng 74HC595: - sendBit8To595(BIT_8[ch+1]) - (bit-bang: set DS, shift 8 bit, strobe ST) 5. Delay 10ms để sensor ổn định 6. Phát TRIG pulse 10µs 7. Đợi ECHO (timeout 40ms = pulseIn) 8. Tính distance = echoTime / 29.1 / 2 9. Lưu vào currentDistances[ch] 10. Tắt relay (BIT_8[0]) 11. Quay lại bước 2 (cảm biến tiếp theo) 12. Sau khi scan 15 cảm biến, mảng currentDistances[15] cập nhật
Luồng Thay Thế	A1: Không MUX - dùng 15 chân TRIG/ECHO riêng biệt (tốn pin) A2: I2C MUX - dùng I2C MUX thay analog MUX A3: Analog multiplexer - dùng CD4067 (khác CD74HC4067)
Luồng Ngoài Lề	E1: MUX selection fail → sai channel, sai distance E2: Shift register bit lỗi → relay không bật/tắt E3: Cảm biến timeout → distance = -1 E4: ECHO không đọc được → pulseIn timeout
Điều Kiện Sau	Mảng currentDistances[15] được cập nhật; Dữ liệu sẵn sàng để gửi Backend

Use Case 3: Backend Lấy Dữ Liệu 15 Cảm Biến (GET /data - Pull Model)

Trường	Nội Dung
Tên Use Case	Backend Lấy Dữ Liệu 15 Cảm Biến từ ESP32 (GET /data)
Mô Tả	Backend định kỳ gọi GET /data endpoint trên ESP32 để lấy dữ liệu 15 cảm biến mới nhất (hoặc trigger quét mới). ESP32 trả JSON: {success, data[15], timestamp}.
Tác Nhân	Backend API Server, ESP32 (WebServer port 80), Frontend

Trường	Nội Dung
Mô Tả Chi Tiết	handleGetData() được gọi khi client GET /data. ESP32 có thể: (1) trả dữ liệu currentDistances[15] cache, hoặc (2) quét 15 cảm biến ngay rồi trả. Backend poll mỗi 30s. JSON response: {success: true, data: [d0, d1, ..., d14], timestamp, channels: 15}.
Điều Kiện Tiên Quyết	ESP32 WebServer chạy; Backend có thể kết nối 192.168.4.5:80; 15 cảm biến đã scan
Luộng Chính	1. Frontend/Backend gọi GET http://192.168.4.5:80/data 2. ESP32 handleGetData() trigger 3. Option A: Trả cached currentDistances[15]: - Tạo JSON payload - {success: true, data: [d0...d14], timestamp: millis()} 4. Option B: Quét ngay (force_reset=true): - Gọi docTatCaCamBien() - Chờ scan xong - Trả JSON với dữ liệu mới 5. Backend parse, lưu vào database 6. Tính occupancy_rate (số slot occupied/15) 7. Trả Frontend status 200
Luộng Thay Thế	A1: Push model - ESP32 push dữ liệu mỗi N giây A2: WebSocket - real-time push via WebSocket
Luộng Ngoài Lề	E1: ESP32 offline (>10s timeout) → Backend 504 E2: Scan quá lâu (>5s) → timeout E3: Data invalid → backend fallback cached data
Điều Kiện Sau	Frontend hiển thị grid 15 slots; Occupancy rate cập nhật

Use Case 4: ESP32 Nhận Lệnh Reset Cảm Biến (POST /detect)

Trường	Nội Dung
Tên Use Case	Reset Cảm Biến - Admin Điều Hướng từ Backend (POST /detect)
Mô Tả	Admin gọi POST /api/parking-slots/reset từ Frontend. Backend gọi POST /detect trên ESP32. ESP32 quét lại 15 cảm biến, xóa nhiễu, trả dữ liệu mới.
Tác Nhân	Admin User, Frontend, Backend API, ESP32
Mô Tả Chi Tiết	handleDetect() gọi docTatCaCamBien() để scan 15 cảm biến. Giống UC-2 nhưng được trigger bởi POST request. Trả JSON: {success: true, message: "Detected 15 sensors", data: [d0...d14]}.

Trường	Nội Dung
Điều Kiện Tiên Quyết	User là Admin; ESP32 WebServer chạy; Cảm biến cần hiệu chỉnh
Luộng Chính	1. Admin Frontend: POST /api/parking-slots/reset 2. Backend kiểm tra admin_required 3. Nếu admin, gọi ESP32Service.reset_sensors() 4. HTTPClient POST http://192.168.4.5:80/detect 5. ESP32 handleDetect() trigger 6. Gọi docTatCaCamBien() quét 15 cảm biến 7. Lưu vào currentDistances[15] 8. Trả JSON: {success: true, message: "Detected 15 sensors", data: [...]} 9. Backend trả 200 OK 10. Frontend: "Reset successful" notification
Luộng Thay Thế	A1: Auto reset - System auto reset nếu detect anomaly A2: Scheduled reset - Reset mỗi N giờ
Luộng Ngoài Lề	E1: User không admin → 403 E2: ESP32 offline → 503 E3: Scan timeout → success: false
Điều Kiện Sau	15 cảm biến quét xong; Dữ liệu mới; Frontend cập nhật

Use Case 5: 74HC595 Shift Register Điều Khiển Relay

Trường	Nội Dung
Tên Use Case	74HC595 Bit-Bang - Điều Khiển Relay/MOSFET cho 15 Cảm Biến
Mô Tả	74HC595 Shift Register (8-bit) được điều khiển bằng bit-bang (bit-by-bit) để bật/tắt relay hoặc MOSFET cấp nguồn cho 15 cảm biến. Dùng 3 chân: DS (data serial), SH (shift clock), ST (strobe).
Tác Nhân	ESP32, 74HC595 Shift Register, Relay/MOSFET
Mô Tả Chi Tiết	Hàm sendBit8To595(byte val) gửi 8-bit val tuần tự qua DS, shift ra bằng SH pulse, strobe bằng ST pulse. Mảng BIT_8[9] chứa bit patterns: BIT_8[0]=all OFF, BIT_8[1..8]=bật relay 1..8 (tương ứng Q0..Q7). Để điều khiển 15 channel, dùng 2 shift register nối cascade (hoặc mở rộng BIT_8 thành uint16_t).

Trường	Nội Dung
Điều Kiện Tiên Quyết	74HC595 kết nối đúng (DS, SH, ST); Relay/MOSFET kết nối Q outputs
Luộng Chính	1. Để bật relay N (N=0..14): - byte pattern = BIT_8[N+1] - sendBit8To595(pattern) 2. sendBit8To595(byte val): - Lặp i = 7 downto 0 - Bit = (val >> i) & 1 - digitalWrite(DS, Bit) - digitalWrite(SH, HIGH) → LOW (shift clock) - delay(1µs) 3. Sau khi shift 8 bit: - digitalWrite(ST, HIGH) → LOW (strobe) - Output Q0..Q7 latch vào giá trị mới 4. Để tắt toàn bộ relay: - sendBit8To595(BIT_8[0]) = 0xFF (all OFF active-LOW) 5. Quay lại bước 1 (relay N tiếp theo)
Luộng Thay Thế	A1: SPI - Dùng SPI thay bit-bang (nhanh hơn) A2: 2x 74HC595 - Cascade 2 register để 16 output
Luộng Ngoài Lề	E1: Bit order sai → relay bật sai E2: Timing sai (delay quá ngắn) → data corrupt E3: Relay không response → check MR (Master Reset), OE
Điều Kiện Sau	Relay được bật/tắt đúng; Cảm biến nhận được nguồn

Use Case 6: CD74HC4067 MUX Chọn Channel Cảm Biến

Trường	Nội Dung
Tên Use Case	CD74HC4067 MUX 16:1 - Chọn Channel ECHO
Mô Tả	CD74HC4067 MUX 16:1 (analog multiplexer) chọn 1 trong 16 channel ECHO từ 15 cảm biến. Điều khiển bằng 4 chân S0-S3 (binary selection: 0..15).
Tác Nhân	ESP32, CD74HC4067 MUX IC

Trường	Nội Dung
Mô Tả Chi Tiết	selectMuxChannel(int ch) set 4 chân S0-S3 thành binary representation của ch (0..15). Pin: S0=GPIO12, S1=GPIO13, S2=GPIO14, S3=GPIO27. Nếu ch=5, set S0=1, S1=0, S2=1, S3=0 (binary 0101). MUX output (pin Y) được nối vào PIN_ECHO=26.
Điều Kiện Tiên Quyết	MUX IC kết nối đúng (S0-S3, Y, GND, VCC); 15 cảm biến nối vào C0..C14
Luộng Chính	1. selectMuxChannel(ch) gọi trước khi phát TRIG: - bit0 = ch & 1 - bit1 = (ch >> 1) & 1 - bit2 = (ch >> 2) & 1 - bit3 = (ch >> 3) & 1 2. digitalWrite(S0, bit0) 3. digitalWrite(S1, bit1) 4. digitalWrite(S2, bit2) 5. digitalWrite(S3, bit3) 6. Delay 1µs để MUX settle 7. Giờ MUX output Y nối với cảm biến ch 8. Phát TRIG pulse 9. Đọc ECHO từ PIN_ECHO (sẽ là ECHO của cảm biến ch) 10. Tính distance
Luộng Thay Thế	A1: I2C MUX - Dùng TCA9548A I2C mux A2: Analog switch - Dùng logic gate NAND/NOR
Luộng Ngoài Lề	E1: Bit order sai → sai channel E2: Settling time quá ngắn → MUX chưa switch E3: MUX out-of-spec voltage → ECHO error
Điều Kiện Sau	Channel N được chọn; ECHO từ cảm biến N được đọc

Use Case 7: ESP32 Xử Lý Lỗi WiFi (Auto Reconnection)

Trường	Nội Dung
Tên Use Case	WiFi Reconnection - Auto Reconnect Logic
Mô Tả	WiFi bị mất (AP down, signal weak). ESP32 tự động phát hiện (mỗi 30s) và reconnect (timeout 10s). Nếu fail, retry sau 30s.
Tác Nhân	ESP32, WiFi Network (UNO R4 AP)

Trường	Nội Dung
Mô Tả Chi Tiết	checkWiFiConnection() gọi mỗi 30s trong loop(). Kiểm tra WiFi.status(). Nếu không connected, gọi reconnectWiFi(). Reconnect timeout 10s. Nếu success, log "Reconnected", nếu fail, retry sau 30s.
Điều Kiện Tiên Quyết	ESP32 loop chạy; WiFi credentials đã config
Luộng Chính	1. Loop() chạy 2. Mỗi 30s gọi checkWiFiConnection() 3. Kiểm tra (now - lastWiFiCheck) >= 30000 4. Nếu WiFi.status() != WL_CONNECTED: - Log "Connection lost" - Gọi reconnectWiFi() - WiFi.disconnect() - delay(1000) - WiFi.config(static IP) - WiFi.begin(ssid, password) - Loop while WiFi not connected, timeout 10s - Nếu success → Log "Reconnected" - Nếu timeout → Log "Reconnect Failed", retry sau 30s 5. Quay lại bước 2
Luộng Thay Thế	A1: Full WiFi.begin() - Complete handshake A2: Manual reset - Admin reset ESP32
Luộng Ngoài Lề	E1: AP down → Reconnect fail, retry E2: Password sai → Reconnect fail liên tục E3: Interference → May disconnect liên tục
Điều Kiện Sau	WiFi reconnect attempt; Nếu success → data flow resume

Use Case 8: Backend Quản Lý Lịch Sử Dữ Liệu (GET /api/parking-slots/history)

Trường	Nội Dung
Tên Use Case	Backend Lấy Lịch Sử Dữ Liệu - Time Series Trend
Mô Tả	Frontend cần hiển thị biểu đồ occupancy trend (24h, 7d, 30d). Backend query cơ sở dữ liệu để lấy lịch sử từ bảng ParkingSlot. Lịch sử được tạo từ mỗi GET /data call từ ESP32.

Trường	Nội Dung
Tác Nhân	Frontend, Backend API, Database (ParkingSlot table)
Mô Tả Chi Tiết	Frontend GET /api/parking-slots/history?hours=24. Backend query DB: SELECT * FROM parking_slot WHERE timestamp >= now() - 24h. Aggregate (group by hour, avg occupancy_rate). Cache 1h. Trả time series. Frontend Chart.js.
Điều Kiện Tiên Quyết	Database lưu ParkingSlot history; Backend endpoint implement; Frontend Chart.js ready
Luộng Chính	1. Frontend "Analytics" → "Occupancy Trend" 2. Chọn timerange: 24h / 7d / 30d 3. Frontend GET /api/parking-slots/history?hours=24 4. Backend cache check (1h TTL) 5. Query DB: SELECT occupancy_rate WHERE timestamp >= now() - 24h 6. Aggregate: GROUP BY hour, AVG(occupancy_rate) 7. Trả time series [{time, rate}, ...] 8. Frontend Chart.js vẽ 9. Hiển thị trend
Luộng Thay Thế	A1: WebSocket real-time A2: Monthly aggregate
Luộng Ngoài Lề	E1: No data → empty array E2: Query slow → timeout, fallback cache E3: DB error → 500
Điều Kiện Sau	Frontend hiển thị trend; User quan sát pattern

Kết Luận

ESP32 là "trái tim cảm biến" quản lý **15 cảm biến HC-SR04** thông qua **74HC595 Shift Register** + **CD74HC4067 MUX**. Kết nối WiFi AP từ UNO R4, cung cấp 2 WebServer endpoint (/data, /detect) cho Backend pull dữ liệu. Lịch sử dữ liệu quản lý bởi Backend database.

Kiến trúc:

- **Hardware:** 15 HC-SR04 + 1 MUX 16:1 + 1 Shift Register + 1 TRIG/ECHO chung
- **Software:** selectMuxChannel() + sendBit8To595() + docTatCaCamBien() = quét 15 cảm biến
- **Network:** WiFi AP + WebServer 2 endpoints + Pull Model
- **Data:** currentDistances[15] cache + API response JSON