KC-Link PRO A8 Technical Manual

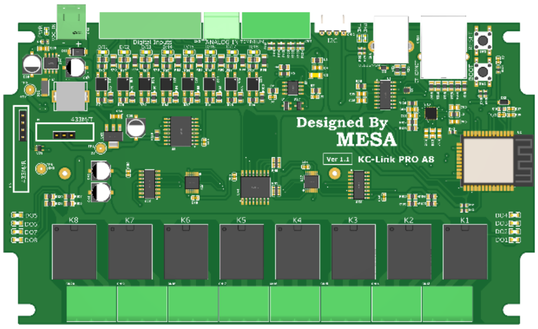


Table of Contents

1. Product Overview
2. Technical Specifications
3. Hardware Components
4. Installation Guide
5. Wiring Instructions
6. Communication Protocols
7. Programming with Arduino
8. Troubleshooting
9. Maintenance
10. Safety Guidelines

Product Overview

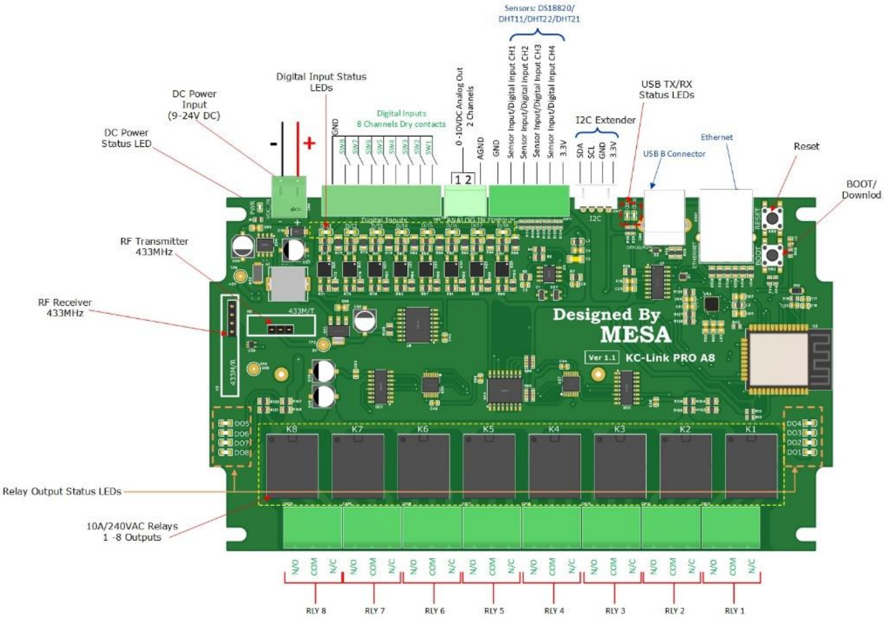
The KC-Link PRO A8 is a professional-grade IoT control board designed by MESA, featuring 8-channel relay outputs and multiple input/output options for advanced automation applications. This versatile development board combines powerful hardware capabilities with flexible connectivity options to support a wide range of industrial and smart home control systems.

The board is based on the ESP32 microcontroller, providing both Wi-Fi and Ethernet connectivity through the LAN8720A chip. With 8 optically isolated digital inputs, 8 high-power relays, 2 analog inputs, and support for multiple temperature/humidity sensors, the KC-Link PRO A8 offers a comprehensive solution for automation projects.

Technical Specifications

Feature	Specification
Microcontroller	ESP32-WROOM-32
Digital Inputs	8 channels (optically isolated)
Relay Outputs	8 channels (10A/240VAC)
Analog Inputs	2 channels (0-5VDC)
Power Supply	9-24V DC
Communication	Wi-Fi, Ethernet, USB, I²C, 433MHz RF
Temperature Sensors	Up to 4 (DS18B20/DHT11/DHT22/DHT21)
Operating Temperature	-10°C to 50°C
Dimensions	Standard DIN rail mountable
Certification	CE, RoHS compliant

Hardware Components



Main Components

- **ESP32 Microcontroller**: Powers all logic operations with built-in Wi-Fi capabilities
- **8 Power Relays (K1-K8)**: Each relay supports 10A/240VAC loads
- **Ethernet Controller**: LAN8720A chip for stable network connectivity
- **CH340C USB Interface**: For programming and serial communication
- **PCF8574 I/O Expanders**: For digital I/O management
- **433MHz RF Modules**: Transmitter and receiver for wireless control
- **I²C Interface**: For connecting external sensors and devices

Input/Output Interfaces

- **Digital Inputs**: 8 optically isolated inputs with status LEDs
- **Analog Inputs**: 2 channels with 0-5VDC range
- **Temperature/Humidity Sensor Ports**: 4 dedicated ports
- **USB Port**: USB B connector for programming and communication
- **Ethernet Port**: RJ45 connector for network connectivity
- **Reset Button**: For system reset
- **Boot/Download Button**: For firmware updates

Installation Guide

Mounting Options

1. **DIN Rail Mounting**: Use standard 35mm DIN rail brackets for cabinet installation
2. **Wall Mounting**: Secure using the mounting holes on the board corners

Power Connection

1. Connect a 9-24V DC power supply to the power input terminals (observe polarity)
2. Verify power by checking the DC Power Status LED

Network Connection

- **Ethernet**: Connect an Ethernet cable to the RJ45 port
- **Wi-Fi**: Configure Wi-Fi settings through the programming interface

Wiring Instructions

Digital Inputs

Connect dry contacts or voltage-free signals to the digital input terminals:

- Each input requires a connection between the corresponding input terminal and GND
- Maximum input voltage: 24V DC
- Digital inputs are optically isolated for protection

Relay Outputs

Each relay provides three connection points:

- **NO (Normally Open)**: Connected to COM when relay is activated
- **COM (Common)**: The common connection point
- **NC (Normally Closed)**: Connected to COM when relay is deactivated

Maximum ratings:

- 10A/240VAC for resistive loads
- 3A/240VAC for inductive loads

Analog Inputs

- Connect analog voltage sources between analog input terminals and GND
- Input range: 0-5V DC
- Resolution: 10-bit (0-1023)

Temperature Sensors

- Connect compatible sensors (DS18B20/DHT11/DHT22/DHT21) to the dedicated ports
- Each port provides power, ground, and data connections

Communication Protocols

Modbus RTU

The KC-Link PRO A8 supports Modbus RTU protocol for industrial integration. The attached document provides detailed protocol information including:

Reading Digital Outputs (Function Code 01)

- Example command: `01 01 00 00 00 40 FA 3D`
- Response format: `01 01 08 [8 bytes data] [CRC]`
- Each bit in the response represents the state of one output channel

Reading Digital Inputs (Function Code 02)

- Example command: `01 02 00 00 00 40 FA 79`
- Response format: `01 02 08 [8 bytes data] [CRC]`
- Each bit in the response represents the state of one input channel

Reading Analog Inputs (Function Code 03)

- Example command: `01 03 00 00 00 04 09 44`
- Response format: `01 03 08 [8 bytes data] [CRC]`
- Every 2 bytes represent one analog channel value

Controlling Digital Outputs

- Individual control (Function code 05):
  - Example (Turn ON output 1): `01 05 00 00 FF 00 8C 3A`
  - Example (Turn OFF output 1): `01 05 00 00 00 00 CD CA`
- Multiple control (Function code 0F):
  - Example: `01 0F 00 00 00 40 08 [8 bytes data] [CRC]`
  - Each bit in the data controls one output channel

MQTT Protocol

For IoT applications, the board can be configured to communicate using MQTT:

- Publish states of inputs and outputs to configurable topics
- Subscribe to command topics for remote control
- Support for automatic discovery in systems like Home Assistant

HTTP/REST API

When configured with appropriate firmware, the board can provide a REST API for control:

- GET requests to read input/output states
- POST requests to control outputs
- JSON response format for easy integration

Programming with Arduino

The KC-Link PRO A8 can be programmed using the Arduino IDE. To get started:

1. Install the Arduino IDE (version 1.8.5 or later)
2. Install ESP32 board support through Boards Manager
3. Select "NodeMCU-32S" from the board list
4. Connect the board via USB and select the correct COM port
5. Install required libraries:
   - PCF8574 (for digital I/O expansion)
   - PubSubClient (for MQTT communication)
   - ArduinoJSON (for API responses)
   - DHT sensor library (for temperature/humidity sensors)

Arduino Examples

Example 1: Basic Relay Control

cpp

```
#include "Arduino.h"
#include "PCF8574.h"

// PCF8574 address for relay control
#define PCF8574_RELAY_ADDR 0x20

// Create PCF8574 instance
PCF8574 relayModule(PCF8574_RELAY_ADDR);

void setup() {
  Serial.begin(115200);
  Serial.println("KC868-A8 Relay Control Example");

  // Initialize Wire for I2C communication
  Wire.begin();

  // Check if PCF8574 is reachable
  if (relayModule.begin()) {
    Serial.println("PCF8574 relay module initialized");
  } else {
    Serial.println("PCF8574 relay module not found");
    while(1);
  }

  // Set all pins as OUTPUT
  for (int i = 0; i < 8; i++) {
    relayModule.pinMode(i, OUTPUT);
    relayModule.digitalWrite(i, HIGH); // Relays are active LOW
  }
}

void loop() {
  // Sequence through all relays
  for (int relay = 0; relay < 8; relay++) {
    // Turn ON the current relay (active LOW)
    Serial.print("Turning ON Relay ");
    Serial.println(relay + 1);
    relayModule.digitalWrite(relay, LOW);
    delay(1000);

    // Turn OFF the current relay
    Serial.print("Turning OFF Relay ");
    Serial.println(relay + 1);
    relayModule.digitalWrite(relay, HIGH);
    delay(500);
  }
}
```

Example 2: Reading Digital Inputs

cpp

```cpp
#include "Arduino.h"
#include "PCF8574.h"

// PCF8574 address for digital inputs
#define PCF8574_INPUT_ADDR 0x22

// Create PCF8574 instance
PCF8574 inputModule(PCF8574_INPUT_ADDR);

// Previous input states for change detection
uint8_t prevInputs = 0;

void setup() {
  Serial.begin(115200);
  Serial.println("KC868-A8 Digital Input Example");

  // Initialize Wire for I2C communication
  Wire.begin();

  // Check if PCF8574 is reachable
  if (inputModule.begin()) {
    Serial.println("PCF8574 input module initialized");
  } else {
    Serial.println("PCF8574 input module not found");
    while(1);
  }

  // Set all pins as INPUT
  for (int i = 0; i < 8; i++) {
    inputModule.pinMode(i, INPUT);
  }

  // Read initial states
  prevInputs = inputModule.read8();
}


void loop() {
  // Read all inputs at once
  uint8_t currentInputs = inputModule.read8();
  // Check if there are changes
  if (currentInputs != prevInputs) {
    Serial.println("Input state change detected:");

    // Check each input
    for (int i = 0; i < 8; i++) {
      bool currentState = bitRead(currentInputs, i);
      bool prevState = bitRead(prevInputs, i);

      if (currentState != prevState) {
        Serial.print("Input ");
        Serial.print(i + 1);
        Serial.print(": ");
        Serial.println(currentState ? "HIGH" : "LOW");
      }
    }

    // Update previous state
    prevInputs = currentInputs;
  }

  delay(100); // Short delay to avoid excessive readings
}
```

Example 3: Reading Analog Inputs

```cpp
#include "Arduino.h"

// Analog input pins (may vary based on board version)
// For version V1.4
#define ANALOG_INPUT_1 34
#define ANALOG_INPUT_2 35

// For older versions
// #define ANALOG_INPUT_1 32
// #define ANALOG_INPUT_2 33

void setup() {
  Serial.begin(115200);
  Serial.println("KC868-A8 Analog Input Example");

  // Configure ADC resolution (0-4095)
  analogReadResolution(12);
}

void loop() {
  // Read analog values
  int analog1 = analogRead(ANALOG_INPUT_1);
  int analog2 = analogRead(ANALOG_INPUT_2);

  // Convert to voltage (0-5V)
  float voltage1 = analog1 * 5.0 / 4095.0;
  float voltage2 = analog2 * 5.0 / 4095.0;

  // Display readings
  Serial.print("Analog Input 1: ");
  Serial.print(analog1);
  Serial.print(" (");
  Serial.print(voltage1, 2);
  Serial.println("V)");

  Serial.print("Analog Input 2: ");
  Serial.print(analog2);
  Serial.print(" (");
  Serial.print(voltage2, 2);
  Serial.println("V)");

  delay(1000);
}
```

Example 4: Temperature Sensor Reading (DS18B20)

cpp

```
#include "Arduino.h"
#include <OneWire.h>
#include <DallasTemperature.h>

// Temperature sensor pins (for V1.4)
// TEMP_SENSOR_1 is connected to GPIO14
#define TEMP_SENSOR_1 14

// Initialize OneWire and DallasTemperature
OneWire oneWire(TEMP_SENSOR_1);
DallasTemperature sensors(&oneWire);

void setup() {
  Serial.begin(115200);
  Serial.println("KC868-A8 Temperature Sensor Example");

  // Start the DS18B20 sensor
  sensors.begin();
}

void loop() {
```

```cpp
  // Request temperature readings
  Serial.println("Requesting temperatures...");
  sensors.requestTemperatures();

  // Read temperature value in Celsius
  float tempC = sensors.getTempCByIndex(0);

  // Check if reading was successful
  if (tempC != DEVICE_DISCONNECTED_C) {
    Serial.print("Temperature: ");
    Serial.print(tempC);
    Serial.println("°C");
  } else {
    Serial.println("Error getting temperature");
  }

  delay(2000);
}
```

Example 5: Web Server for Remote Control

cpp

```cpp
#include "Arduino.h"
#include "PCF8574.h"
#include <WiFi.h>
#include <WebServer.h>
#include <ArduinoJson.h>

// Network credentials
const char* ssid = "YourNetworkName";
const char* password = "YourPassword";

// PCF8574 address for relay control
#define PCF8574_RELAY_ADDR 0x20

// Create PCF8574 instance
PCF8574 relayModule(PCF8574_RELAY_ADDR);

// Create web server on port 80
WebServer server(80);

void setup() {
  Serial.begin(115200);
  Serial.println("KC868-A8 Web Server Example");

  // Initialize Wire for I2C communication
  Wire.begin();
  // Check if PCF8574 is reachable
  if (relayModule.begin()) {
    Serial.println("PCF8574 relay module initialized");
  } else {
    Serial.println("PCF8574 relay module not found");
    while(1);
  }

  // Set all pins as OUTPUT and turn OFF all relays
  for (int i = 0; i < 8; i++) {
    relayModule.pinMode(i, OUTPUT);
    relayModule.digitalWrite(i, HIGH); // Relays are active LOW
  }

  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");}
  Serial.println();
  Serial.print("Connected to WiFi. IP Address: ");
  Serial.println(WiFi.localIP());

  // Set up web server routes
  server.on("/", HTTP_GET, handleRoot);
  server.on("/api/relay", HTTP_GET, handleGetRelays);
  server.on("/api/relay", HTTP_POST, handleSetRelay);
  server.onNotFound(handleNotFound);

  // Start server
  server.begin();
  Serial.println("HTTP server started");
}

void loop() {
  // Handle client requests
  server.handleClient();
}

// Root page handler
void handleRoot() {
  String html = "<html><head><title>KC868-A8 Control</title>";
  html += "<meta name='viewport' content='width=device-width, initial-scale=1'>";
  html += "<style>body{font-family:Arial;margin:20px;}";
  html += ".relay{background-color:#4CAF50;border:none;color:white;padding:15px 32px;";
  html += "text-align:center;text-decoration:none;display:inline-block;font-size:16px;";
  html += "margin:4px 2px;cursor:pointer;border-radius:4px;width:200px;}";
  html += ".relay.off{background-color:#f44336;}</style></head>";
  html += "<body><h1>KC868-A8 Relay Control</h1>";

  // Create buttons for each relay
  for (int i = 0; i < 8; i++) {
    bool relayState = !relayModule.digitalRead(i); // Inverted because relays are active LOW
    String buttonClass = relayState ? "relay" : "relay off";
    String buttonText = "Relay " + String(i + 1) + ": " + (relayState ? "ON" : "OFF");

    html += "<button class='" + buttonClass + "' onclick='toggleRelay(" + String(i) + ")'>" + b
  }

  // Add JavaScript for AJAX control
  html += "<script>";
  html += "function toggleRelay(relay) {";
  html += "  var xhr = new XMLHttpRequest();";
  html += "  xhr.open('POST', '/api/relay', true);";
  html += "  xhr.setRequestHeader('Content-Type', 'application/json');";
  html += "  xhr.onreadystatechange = function() {";
  html += "    if (xhr.readyState == 4 && xhr.status == 200) {";
  html += "      location.reload();";
  html += "    }";
  html += "  };";
  html += "  var data = JSON.stringify({relay: relay, state: 'toggle'});";
  html += "  xhr.send(data);";
  html += "}";
  html += "</script></body></html>";

  server.send(200, "text/html", html);
}

// API endpoint to get all relay states
void handleGetRelays() {
  DynamicJsonDocument doc(200);
  JsonArray relays = doc.createNestedArray("relays");

  for (int i = 0; i < 8; i++) {
    bool relayState = !relayModule.digitalRead(i); // Inverted because relays are active LOW
    relays.add(relayState);
  }

  String response;
  serializeJson(doc, response);
  server.send(200, "application/json", response);
}

// API endpoint to set relay state
```

```
void handleSetRelay() {
  String body = server.arg("plain");
  DynamicJsonDocument doc(200);
  DeserializationError error = deserializeJson(doc, body);

  if (error) {
    server.send(400, "application/json", "{\"status\":\"error\",\"message\":\"Invalid JSON\"}")
    return;
  }

  // Check for required parameters
  if (!doc.containsKey("relay") || !doc.containsKey("state")) {
    server.send(400, "application/json", "{\"status\":\"error\",\"message\":\"Missing relay or
    return;
  }

  int relay = doc["relay"];
  String state = doc["state"];
  // Validate relay number
  if (relay < 0 || relay > 7) {
    server.send(400, "application/json", "{\"status\":\"error\",\"message\":\"Invalid relay num
    return;
  }

  // Set relay state
  bool currentState = !relayModule.digitalRead(relay); // Inverted because relays are active LO
  bool newState;

  if (state == "on" || state == "ON" || state == "1") {
    newState = true;
  } else if (state == "off" || state == "OFF" || state == "0") {
    newState = false;
  } else if (state == "toggle") {
    newState = !currentState;
  } else {
    server.send(400, "application/json", "{\"status\":\"error\",\"message\":\"Invalid state val
    return;
  }

  // Update relay
  relayModule.digitalWrite(relay, !newState); // Inverted because relays are active LOW

  // Send response
  DynamicJsonDocument response(200);
  response["status"] = "success";
  response["relay"] = relay;
  response["state"] = newState ? "on" : "off";
  String jsonResponse;
  serializeJson(response, jsonResponse);
  server.send(200, "application/json", jsonResponse);
}

// Not found handler
void handleNotFound() {
  server.send(404, "text/plain", "Not found");
}
```

Troubleshooting

LED Indicators

- **Power LED**: Indicates proper power supply
- **Digital Input Status LEDs**: Show state of each digital input
- **Relay Output Status LEDs**: Show state of each relay
- **USB TX/RX Status LEDs**: Flash during serial communication
- **Ethernet Status LEDs**: Indicate network connection and activity

Common Issues and Solutions

Issue	**Possible Cause**	**Solution**
No power LED	Power supply disconnected or faulty	Check power connections and voltage
Relay doesn't activate	Command issue or relay failure	Test relay manually, check command format
Cannot connect via USB	Driver issue or cable problem	Install correct CH340 drivers, try different cable
Ethernet not working	Network configuration issue	Check cable, network settings, and IP address
Input not detected	Incorrect wiring or input voltage	Verify wiring and voltage levels
Sensors not reading	Incompatible sensor or wiring issue	Check sensor compatibility and connections
I2C communication fails	Address conflict or wiring issue	Verify I2C addresses and connections
PCF8574 not responding	Wrong I2C address or hardware issue	Check I2C addresses (0x20 for relays, 0x22 for inputs)

Maintenance

Regular Maintenance Checks

- Inspect terminal connections for tightness
- Check relay contacts for wear or pitting
- Verify cooling vents are unobstructed
- Update firmware to latest version for improved functionality

Relay Lifespan

- Mechanical lifespan: Approximately 100,000 operations
- Electrical lifespan: Varies based on load type and switching frequency
- Recommendation: For critical applications, implement a rotation strategy for loads

Safety Guidelines

Installation Safety

- Installation should be performed by qualified personnel
- Always disconnect power before making any connections
- Use appropriate wire gauges for the current requirements
- Install in a ventilated enclosure protected from moisture and dust

Operational Safety

- Do not exceed the maximum ratings for inputs and outputs
- Maintain separation between signal wiring and power wiring
- Use suppression devices for inductive loads to protect relay contacts
- Implement additional protection for controlling critical equipment

Regulatory Compliance

- The board is designed to comply with relevant electrical safety standards
- Installation must comply with local electrical codes and regulations
- For industrial applications, additional isolation may be required

---

For additional support and resources, please visit the official MESA website or the KinCony support forum. This technical manual is subject to updates as firmware and hardware improvements are released.