**CORTEX LINK A8F-M**

**TECHNICAL USER MANUAL**

A green circuit board with many small chips

AI-generated content may be incorrect. A green circuit board with many small components

AI-generated content may be incorrect.

**Designed and Manufactured by MESA**  
*Microcode Embedded Systems and Automation*

**TABLE OF CONTENTS**

1. [Introduction](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#1-introduction)
2. [Getting Started](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#2-getting-started)
3. [Hardware Overview](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#3-hardware-overview)
4. [Power Requirements](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#4-power-requirements)
5. [Input/Output Interfaces](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#5-inputoutput-interfaces)
6. [Communication Interfaces](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#6-communication-interfaces)
7. [Programming & Development](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#7-programming--development)
8. [Integration Examples](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#8-integration-examples)
9. [Troubleshooting](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#9-troubleshooting)
10. [Technical Reference](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#10-technical-reference)
11. [Appendices](https://claude.ai/chat/3a73d54d-ce97-4e70-9e90-78dc39ed2cf6#11-appendices)

**1. INTRODUCTION**

**1.1 Product Overview**

The Cortex Link A8F-M ESP32 is a high-performance IoT controller board designed for versatile automation applications in both residential and industrial environments. Built around the powerful ESP32 dual-core processor, this controller offers extensive I/O capabilities, multiple communication interfaces, and seamless integration with popular development platforms.

**1.2 Key Features**

* **Powerful Processing**: ESP32 dual-core 32-bit processor @ 240MHz with 8MB flash memory
* **Extensive I/O**: 16 MOSFET outputs, 8 digital inputs, 4 analog inputs, 2 analog outputs
* **Versatile Connectivity**: Wi-Fi, Bluetooth, Ethernet, RS485/Modbus, optional GSM
* **Expansion Capability**: I2C interface, 1-Wire support, RF transmitter/receiver options
* **Integration-Ready**: Compatible with Arduino IDE, ESPHome, Home Assistant, MicroPython
* **Industrial-Grade Design**: Operating temperature -40°C to +85°C, CE certified, RoHS compliant

**1.3 Applications**

* **Smart Home Automation**: Lighting control, HVAC management, security systems
* **Industrial Control**: Equipment monitoring, process automation, data collection
* **IoT Solutions**: Remote monitoring, telemetry, sensor networks
* **Building Management**: Energy optimization, environmental control, access systems

**1.4 Package Contents**

* 1 × Cortex Link A8F-M ESP32 Board
* 1 × USB Cable (Type B)
* User Manual (digital download)

**1.5 Product Availability**

This product can be manufactured in any quantity upon request. For inquiries, bulk orders, or customization options, contact MESA directly via email or through our website.

**2. GETTING STARTED**

**2.1 Initial Setup**

1. **Unbox and Inspect**: Carefully remove the board from packaging and inspect for any shipping damage
2. **Mount the Board**: Secure the board in its intended location using the mounting holes
3. **Connect Power**: Apply 9-12V DC to the power input terminals (observe polarity)
4. **Establish Communication**: Connect to the board via USB, Ethernet, or Wi-Fi

**2.2 First-Time Configuration**

**2.2.1 USB Connection**

1. Connect the USB cable to the board's USB-B port and your computer
2. Install appropriate USB drivers if needed
3. Open your serial terminal application (115200 baud, 8N1)
4. Press the RESET button to verify communication

**2.2.2 Network Configuration**

The default network configuration depends on your programming method:

**For Arduino IDE projects:**

* Upload a sketch that configures Wi-Fi credentials
* Monitor the serial output for the assigned IP address

**For ESPHome projects:**

* Configure network settings in the YAML file
* Upload the configuration via USB
* The device will connect to the specified network

**2.3 Quick Test Procedure**

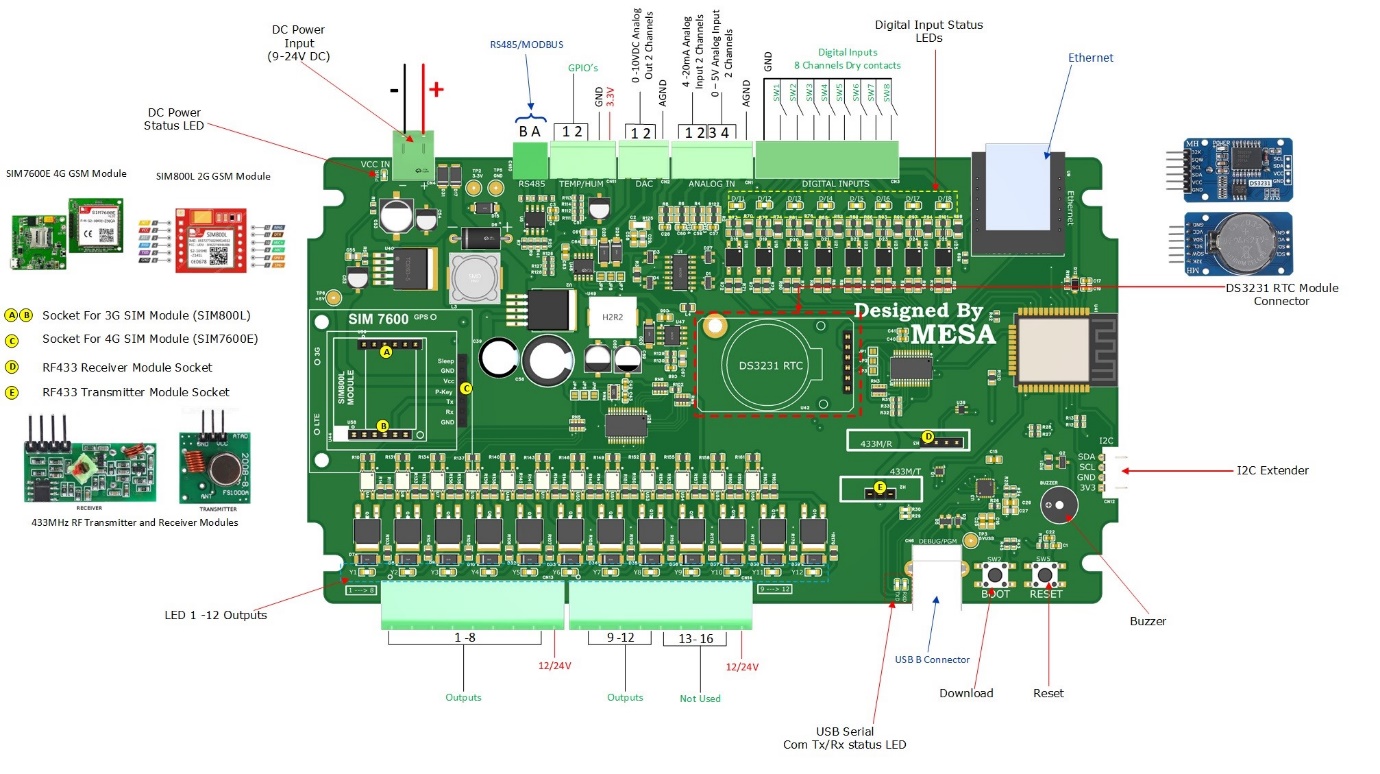
To verify your board is functioning correctly:

1. **Power Test**: Confirm the power LED illuminates when powered
2. **Communication Test**: Establish serial communication via USB
3. **I/O Test**: Run a basic test sketch to verify inputs and outputs
4. **Network Test**: Confirm connectivity via Wi-Fi or Ethernet

**3. HARDWARE OVERVIEW**

**3.1 Board Layout**

The Cortex Link A8F-M ESP32 board is organized into functional sections:



1. **Power Section**: Power input terminals and regulation circuitry
2. **Communication Interfaces**: RS485, Ethernet, GSM socket, RF module sockets
3. **Digital Input Section**: 8 optically isolated inputs with status LEDs
4. **Analog Input Section**: 4-20mA and 0-5V analog input terminals
5. **Output Section**: 16 MOSFET output channels with status LEDs
6. **ESP32 Module**: Main microcontroller with supporting components
7. **Programming Interface**: USB connector and programming buttons
8. **Expansion Interfaces**: I2C, 1-Wire, temperature/humidity sensor connections

**3.2 Indicator LEDs**

The board features multiple status LEDs:

* **Power LED**: Indicates power supply is connected and functional
* **Digital Input LEDs**: Show the status of each digital input
* **Output Status LEDs**: Indicate which outputs are active
* **Communication LEDs**: Display activity on communication interfaces
* **Status LEDs**: Show system status and programming mode

**3.3 Connectors and Terminals**

| **Connector** | **Description** | **Location** |
| --- | --- | --- |
| CN1 | Analog Input Terminal Block (4-20mA & 0-5V) | Center-top of board |
| CN2 | Analog Output Terminal Block (0-10V) | Center-top of board |
| CN3 | Digital Input Terminal Block | Upper-right section |
| CN11 | Temperature/Humidity Sensor Connection | Upper-middle section |
| RS485 | Modbus RTU Communication Terminal | Upper-left section |
| Ethernet | RJ45 Network Connection | Right side |
| USB-B | Programming and Debug Interface | Bottom-right corner |
| Output Terminals | MOSFET Output Connections (1-16) | Bottom section |
| Power Input | 9-12V DC Power Connection | Upper-left corner |

**3.4 Control Buttons**

* **RESET**: Restarts the ESP32 microcontroller
* **BOOT**: Used to enter programming mode (hold while pressing RESET)

**4. POWER REQUIREMENTS**

**4.1 Power Supply Specifications**

* **Input Voltage**: 9V-12V DC
* **Current Requirements**:
  + Idle: ~100mA
  + All outputs active: ~1A (plus connected load current)
* **Power Connector**: 2-pin terminal block (observe polarity)
* **Protection**: Reverse polarity protection, overvoltage protection

**4.2 Power Distribution**

The board features multiple power rails:

* **12V Rail**: Powers the MOSFET outputs and analog circuitry
* **5V Rail**: Logic voltage for various components
* **3.3V Rail**: ESP32 and digital logic components

**4.3 MOSFET Output Power**

* **Output Voltage**: Follows the input power supply (9-12V typical, 24V maximum)
* **Current Rating**: 500mA per channel maximum
* **Isolation**: Outputs are isolated from microcontroller logic

**4.4 Power Considerations**

* Use a regulated power supply with sufficient current capacity
* Keep power wiring separate from signal wiring to reduce interference
* For high-power applications, consider additional external power distribution

**5. INPUT/OUTPUT INTERFACES**

**5.1 Digital Inputs**

The board features 8 optically isolated digital inputs:

* **Input Type**: Dry N/O (normally open) contacts
* **Isolation**: Optical isolation for noise immunity and ground fault protection
* **Indication**: LED status indicator for each input
* **Terminal**: CN3 terminal block
* **Addressing**: Controlled via MCP23017 I/O expander (U8)

**5.1.1 Digital Input Wiring**

Connect dry contact switches or sensors as follows:

1. Connect one side of the contact to the input terminal
2. Connect the other side to the GND terminal
3. When the contact closes, the input activates (active low)

**5.2 Analog Inputs**

The board provides 4 analog input channels:

* **Channels 1-2**: 4-20mA current loop inputs
* **Channels 3-4**: 0-5V voltage inputs
* **Resolution**: 12-bit ADC (4096 steps)
* **Terminal**: CN1 terminal block
* **Direct Connection**: These inputs connect directly to the ESP32 ADC pins

**5.2.1 Analog Input Wiring**

For 4-20mA sensors:

1. Connect the positive lead to the appropriate input terminal
2. Connect the negative lead to the corresponding GND terminal
3. Ensure the sensor is powered appropriately

For 0-5V sensors:

1. Connect the sensor output to the appropriate input terminal
2. Connect the sensor ground to the GND terminal
3. Keep cable lengths short to minimize noise

**5.3 MOSFET Outputs**

The board features 16 MOSFET output channels:

* **Output Type**: N-channel MOSFET low-side switch
* **Voltage Rating**: 12/24V DC maximum
* **Current Rating**: 500mA per channel
* **Control**: Via MCP23017 I/O expanders (U26)
* **Status**: LED indicators for each output

**5.3.1 MOSFET Output Wiring**

1. Connect the positive side of the load to the positive power supply
2. Connect the negative side of the load to the desired output terminal
3. The output activates the load by connecting it to ground

For inductive loads (relays, solenoids, motors):

* Use external flyback diodes to protect the MOSFET outputs
* Consider using external relays for higher current loads

**5.4 Analog Outputs**

The board provides 2 analog output channels:

* **Output Range**: 0-10V DC
* **Resolution**: 12-bit DAC
* **Control**: Via GP8413 DAC (U46)
* **Terminal**: CN2 terminal block

**5.4.1 Analog Output Wiring**

1. Connect the VOUT terminal to the controlled device's input
2. Connect the GND terminal to the controlled device's ground
3. Keep cable lengths short and use shielded cable for noise-sensitive applications

**6. COMMUNICATION INTERFACES**

**6.1 Wi-Fi**

The ESP32 includes integrated Wi-Fi capabilities:

* **Standard**: IEEE 802.11 b/g/n
* **Frequency**: 2.4 GHz
* **Security**: WPA/WPA2/WPA3
* **Modes**: Station mode, Access Point mode, or both
* **Antenna**: Internal PCB antenna with option for external antenna

**6.1.1 Wi-Fi Configuration**

**Arduino IDE Example:**

#include <WiFi.h>

const char\* ssid = "YourNetworkName";

const char\* password = "YourPassword";

void setup() {

Serial.begin(115200);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: " + WiFi.localIP().toString());

}

**ESPHome Example:**

wifi:

ssid: "YourNetworkName"

password: "YourPassword"

# Optional fallback AP

ap:

ssid: "CortexLink Fallback AP"

password: "fallbackpassword"

**6.2 Ethernet**

The board includes a W5500-based Ethernet interface:

* **Connector**: RJ45
* **Speed**: 10/100 Mbps
* **Configuration**: Automatic via DHCP or static IP
* **Control**: SPI interface from ESP32

**6.2.1 Ethernet Configuration**

#include <SPI.h>

#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

void setup() {

Serial.begin(115200);

Serial.println("Initializing Ethernet...");

Ethernet.init(5); // CS pin connected to GPIO5

if (Ethernet.begin(mac) == 0) {

Serial.println("Failed to configure Ethernet using DHCP");

// Try to configure using static IP if DHCP failed

IPAddress ip(192, 168, 1, 177);

IPAddress gateway(192, 168, 1, 1);

IPAddress subnet(255, 255, 255, 0);

IPAddress dns(8, 8, 8, 8);

Ethernet.begin(mac, ip, dns, gateway, subnet);

}

Serial.print("IP address: ");

Serial.println(Ethernet.localIP());

}

**6.3 RS485/Modbus RTU**

The board includes an RS485 interface for Modbus RTU communication:

* **Interface**: Half-duplex RS485
* **Controller**: MAX485 transceiver
* **Connections**: Terminal block with A, B terminals
* **Control**: DE/RE control via ESP32 GPIO27

**6.3.1 RS485/Modbus Configuration**

#include <ModbusMaster.h>

#define MAX485\_DE 27

#define MAX485\_RE 27

#define RX\_PIN 16

#define TX\_PIN 17

ModbusMaster node;

void preTransmission() {

digitalWrite(MAX485\_DE, 1);

digitalWrite(MAX485\_RE, 1);

}

void postTransmission() {

digitalWrite(MAX485\_DE, 0);

digitalWrite(MAX485\_RE, 0);

}

void setup() {

pinMode(MAX485\_DE, OUTPUT);

pinMode(MAX485\_RE, OUTPUT);

digitalWrite(MAX485\_DE, 0);

digitalWrite(MAX485\_RE, 0);

Serial2.begin(9600, SERIAL\_8N1, RX\_PIN, TX\_PIN);

node.begin(1, Serial2);

node.preTransmission(preTransmission);

node.postTransmission(postTransmission);

}

void loop() {

// Read 10 registers starting at 0x00

uint8\_t result = node.readHoldingRegisters(0x00, 10);

if (result == node.ku8MBSuccess) {

Serial.println("Read successful");

for (int i = 0; i < 10; i++) {

Serial.print("Register ");

Serial.print(i);

Serial.print(": ");

Serial.println(node.getResponseBuffer(i));

}

}

delay(1000);

}

**6.4 Bluetooth**

The ESP32 includes integrated Bluetooth capabilities:

* **Version**: Bluetooth 4.2 BR/EDR and BLE (Bluetooth Low Energy)
* **Applications**: Configuration, control, data transfer
* **Range**: Approximately 10 meters (33 feet)

**6.5 GSM (Optional)**

The board provides sockets for optional GSM modules:

* **Module Options**:
  + SIM800L (2G GSM/GPRS) - Socket A
  + SIM7600E (4G LTE) - Socket C
* **Connections**: Dedicated TX/RX pins to ESP32
* **Control**: Power and reset control via GPIO pins

**6.5.1 GSM Module Setup**

1. Insert the appropriate SIM module into the designated socket
2. Install an activated SIM card with data plan
3. Connect the GSM antenna to the module
4. Configure the module with your cellular provider's APN settings

**6.5.2 GSM Code Example**

#include <TinyGSM.h>

#define TINY\_GSM\_MODEM\_SIM800

#define SerialAT Serial1

const char\* apn = "internet"; // Your provider's APN

const char\* user = ""; // APN username (if required)

const char\* pass = ""; // APN password (if required)

TinyGsm modem(SerialAT);

void setup() {

Serial.begin(115200);

SerialAT.begin(115200, SERIAL\_8N1, 26, 25); // RX=GPIO26, TX=GPIO25

Serial.println("Initializing modem...");

modem.restart();

String modemInfo = modem.getModemInfo();

Serial.print("Modem: ");

Serial.println(modemInfo);

Serial.print("Waiting for network...");

if (!modem.waitForNetwork()) {

Serial.println(" fail");

return;

}

Serial.println(" OK");

Serial.print("Connecting to ");

Serial.print(apn);

if (!modem.gprsConnect(apn, user, pass)) {

Serial.println(" fail");

return;

}

Serial.println(" OK");

Serial.print("GPRS status: ");

Serial.println(modem.isGprsConnected() ? "connected" : "not connected");

Serial.print("Signal quality: ");

Serial.println(modem.getSignalQuality());

}

**6.6 RF 433MHz**

The board includes sockets for RF 433MHz communication:

* **Transmitter Socket**: Marked "E" on the board
* **Receiver Socket**: Marked "D" on the board
* **Control**: Dedicated GPIO pins for TX and RX
* **Applications**: Remote control, sensor communication, home automation

**6.6.1 RF Module Code Example**

#include <RCSwitch.h>

#define RF\_TX\_PIN 32

#define RF\_RX\_PIN 33

RCSwitch rfSwitch = RCSwitch();

void setup() {

Serial.begin(115200);

// Configure transmitter

rfSwitch.enableTransmit(RF\_TX\_PIN);

// Configure receiver

rfSwitch.enableReceive(RF\_RX\_PIN);

Serial.println("RF 433MHz initialized");

}

void loop() {

// Transmit example

rfSwitch.send(12345, 24); // Send value "12345" with 24 bit protocol

Serial.println("Transmitted: 12345");

delay(1000);

// Receive example

if (rfSwitch.available()) {

unsigned long value = rfSwitch.getReceivedValue();

Serial.print("Received: ");

Serial.println(value);

rfSwitch.resetAvailable();

}

delay(1000);

}

**7. PROGRAMMING & DEVELOPMENT**

**7.1 Development Environment Setup**

**7.1.1 Arduino IDE**

1. **Install Arduino IDE**: Download from [arduino.cc](https://www.arduino.cc/en/software)
2. **Add ESP32 Board Support**:
   * Go to File > Preferences
   * Add https://dl.espressif.com/dl/package\_esp32\_index.json to "Additional Boards Manager URLs"
   * Go to Tools > Board > Boards Manager
   * Search for "ESP32" and install the package
3. **Select Board Configuration**:
   * Board: "ESP32 Dev Module"
   * Upload Speed: 921600
   * Flash Frequency: 80MHz
   * Flash Mode: QIO
   * Flash Size: 8MB
   * Partition Scheme: Default 4MB with spiffs
4. **Install Required Libraries**:
   * Adafruit MCP23017 Arduino Library
   * ModbusMaster
   * RCSwitch
   * Other libraries as needed for your project

**7.1.2 ESPHome**

1. **Install ESPHome**:
2. pip install esphome
3. **Create Basic Configuration**: Create a YAML file with your device configuration
4. **Compile and Upload**:
5. esphome run your\_config.yaml

**7.1.3 PlatformIO**

1. **Install Visual Studio Code**: Download from [code.visualstudio.com](https://code.visualstudio.com/)
2. **Install PlatformIO Extension**: Search for "PlatformIO" in the Extensions marketplace
3. **Create New Project**:
   * Select ESP32 platform
   * Choose appropriate board (ESP32 Dev Module)
   * Configure framework (Arduino or ESP-IDF)

**7.2 Programming Methods**

**7.2.1 USB Programming**

1. Connect the USB cable to the board's USB-B port
2. Hold the BOOT button while pressing the RESET button to enter programming mode
3. Release both buttons
4. Select the correct COM port in your development environment
5. Upload your code

**7.2.2 Over-the-Air (OTA) Programming**

After initial setup via USB, you can program the board wirelessly:

**Arduino OTA Example:**

#include <WiFi.h>

#include <ArduinoOTA.h>

const char\* ssid = "YourNetworkName";

const char\* password = "YourPassword";

void setup() {

Serial.begin(115200);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

ArduinoOTA.setHostname("cortexlink");

ArduinoOTA.setPassword("otapassword");

ArduinoOTA.onStart([]() {

Serial.println("OTA update starting");

});

ArduinoOTA.onEnd([]() {

Serial.println("OTA update complete");

});

ArduinoOTA.onError([](ota\_error\_t error) {

Serial.printf("OTA Error[%u]: ", error);

if (error == OTA\_AUTH\_ERROR) Serial.println("Auth Failed");

else if (error == OTA\_BEGIN\_ERROR) Serial.println("Begin Failed");

else if (error == OTA\_CONNECT\_ERROR) Serial.println("Connect Failed");

else if (error == OTA\_RECEIVE\_ERROR) Serial.println("Receive Failed");

else if (error == OTA\_END\_ERROR) Serial.println("End Failed");

});

ArduinoOTA.begin();

Serial.println("OTA ready");

}

void loop() {

ArduinoOTA.handle();

// Your code here

}

**ESPHome OTA:** Simply include the OTA component in your YAML:

ota:

password: "otapassword"

**7.3 Home Assistant Integration**

**7.3.1 ESPHome Integration**

1. Build your configuration with ESPHome
2. Include the API component:
3. api: password: "your\_api\_password"
4. Flash the device
5. In Home Assistant, go to Configuration > Integrations
6. Click "+" and select "ESPHome"
7. Enter the device's IP address and API password

**7.3.2 MQTT Integration**

1. Configure the device to connect to your MQTT broker:
2. #include <WiFi.h>
3. #include <PubSubClient.h>
4. WiFiClient espClient;
5. PubSubClient client(espClient);
6. void setup() {
7. // Connect to Wi-Fi (code omitted)
9. client.setServer("192.168.1.100", 1883);
10. client.setCallback(callback);
11. }
12. void loop() {
13. if (!client.connected()) {
14. reconnect();
15. }
16. client.loop();
18. // Publish data
19. client.publish("home/sensor/temperature", "23.5");
20. }
21. void callback(char\* topic, byte\* payload, unsigned int length) {
22. // Handle incoming messages
23. }
24. void reconnect() {
25. while (!client.connected()) {
26. if (client.connect("CortexLinkClient")) {
27. client.subscribe("home/control/#");
28. } else {
29. delay(5000);
30. }
31. }
32. }
33. In Home Assistant, configure MQTT entities:
34. sensor:
35. - platform: mqtt
36. name: "Living Room Temperature"
37. state\_topic: "home/sensor/temperature"
38. unit\_of\_measurement: "°C"

**7.4 MicroPython Support**

The Cortex Link A8F-M ESP32 supports MicroPython:

1. **Flash MicroPython Firmware**:
2. esptool.py --port COM3 erase\_flash
3. esptool.py --port COM3 --baud 460800 write\_flash -z 0x1000 esp32-20220117-v1.18.bin
4. **Upload MicroPython Scripts**: Use tools like Thonny IDE, rshell, or ampy to upload scripts
5. **MicroPython I/O Example**:
6. from machine import Pin, I2C
7. import time
8. # Initialize I2C
9. i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=100000)
10. # Scan for I2C devices
11. devices = i2c.scan()
12. print("I2C devices found:", [hex(device) for device in devices])
13. # Example using onboard LED
14. led = Pin(2, Pin.OUT)
15. while True:
16. led.value(1)
17. time.sleep(0.5)
18. led.value(0)
19. time.sleep(0.5)

**8. INTEGRATION EXAMPLES**

**8.1 Smart Home Automation**

**8.1.1 Home Assistant Integration with ESPHome**

# ESPHome configuration for smart home control

esphome:

name: cortexlink

platform: ESP32

board: esp32dev

wifi:

ssid: "YourWiFiSSID"

password: "YourWiFiPassword"

api:

password: "your\_api\_password"

ota:

password: "your\_ota\_password"

i2c:

sda: 21

scl: 22

scan: true

# Input configuration

mcp23017:

- id: input\_expander

address: 0x21

- id: output\_expander

address: 0x20

binary\_sensor:

- platform: gpio

pin:

mcp23017: input\_expander

number: 0

inverted: true

name: "Motion Sensor"

device\_class: motion

- platform: gpio

pin:

mcp23017: input\_expander

number: 1

inverted: true

name: "Door Sensor"

device\_class: door

# Output configuration

switch:

- platform: gpio

pin:

mcp23017: output\_expander

number: 0

name: "Living Room Lights"

- platform: gpio

pin:

mcp23017: output\_expander

number: 1

name: "Kitchen Lights"

# Analog sensors

sensor:

- platform: adc

pin: GPIO36

name: "Living Room Temperature"

update\_interval: 60s

filters:

- lambda: return (x \* 3.3 / 4095.0) \* 100.0;

unit\_of\_measurement: "°C"

accuracy\_decimals: 1

**8.1.2 Smart Lighting Control System**

**Arduino Code:**

#include <WiFi.h>

#include <PubSubClient.h>

#include <Wire.h>

#include <Adafruit\_MCP23017.h>

// Network and MQTT settings

const char\* ssid = "YourWiFiSSID";

const char\* password = "YourWiFiPassword";

const char\* mqtt\_server = "192.168.1.100";

// I/O setup

Adafruit\_MCP23017 outputExpander;

Adafruit\_MCP23017 inputExpander;

WiFiClient espClient;

PubSubClient client(espClient);

// Input and output mappings

const uint8\_t MOTION\_SENSOR\_1 = 0;

const uint8\_t MOTION\_SENSOR\_2 = 1;

const uint8\_t LIGHT\_SWITCH\_1 = 2;

const uint8\_t LIGHT\_SWITCH\_2 = 3;

const uint8\_t LIGHT\_OUTPUT\_1 = 0;

const uint8\_t LIGHT\_OUTPUT\_2 = 1;

const uint8\_t LIGHT\_OUTPUT\_3 = 2;

// Timing variables

unsigned long lastMotionTime = 0;

const unsigned long AUTO\_OFF\_DELAY = 300000; // 5 minutes

void setup() {

Serial.begin(115200);

// Initialize I/O expanders

Wire.begin();

outputExpander.begin(0x20);

inputExpander.begin(0x21);

// Configure inputs and outputs

for (uint8\_t i = 0; i < 8; i++) {

inputExpander.pinMode(i, INPUT);

inputExpander.pullUp(i, HIGH);

}

for (uint8\_t i = 0; i < 8; i++) {

outputExpander.pinMode(i, OUTPUT);

outputExpander.digitalWrite(i, LOW);

}

// Connect to Wi-Fi

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

}

// Connect to MQTT

client.setServer(mqtt\_server, 1883);

client.setCallback(callback);

}

void callback(char\* topic, byte\* payload, unsigned int length) {

String message = "";

for (int i = 0; i < length; i++) {

message += (char)payload[i];

}

// Handle light control commands

if (String(topic) == "home/lights/living") {

if (message == "ON") {

outputExpander.digitalWrite(LIGHT\_OUTPUT\_1, HIGH);

client.publish("home/lights/living/status", "ON");

} else if (message == "OFF") {

outputExpander.digitalWrite(LIGHT\_OUTPUT\_1, LOW);

client.publish("home/lights/living/status", "OFF");

}

}

// Add more topics as needed

}

void reconnect() {

while (!client.connected()) {

if (client.connect("CortexLinkClient")) {

// Subscribe to control topics

client.subscribe("home/lights/#");

client.subscribe("home/system/#");

} else {

delay(5000);

}

}

}

void loop() {

if (!client.connected()) {

reconnect();

}

client.loop();

// Check motion sensors

bool motion1 = !inputExpander.digitalRead(MOTION\_SENSOR\_1);

bool motion2 = !inputExpander.digitalRead(MOTION\_SENSOR\_2);

if (motion1 || motion2) {

// Motion detected

lastMotionTime = millis();

// Turn on lights if they're not already on

if (outputExpander.digitalRead(LIGHT\_OUTPUT\_1) == LOW) {

outputExpander.digitalWrite(LIGHT\_OUTPUT\_1, HIGH);

client.publish("home/lights/living/status", "ON");

}

} else if (millis() - lastMotionTime > AUTO\_OFF\_DELAY) {

// No motion for delay period, turn off lights

outputExpander.digitalWrite(LIGHT\_OUTPUT\_1, LOW);

client.publish("home/lights/living/status", "OFF");

}

// Check manual switches

static bool prevSwitch1 = false;

bool switch1 = !inputExpander.digitalRead(LIGHT\_SWITCH\_1);

if (switch1 != prevSwitch1) {

if (switch1) {

// Toggle light state

bool currentState = outputExpander.digitalRead(LIGHT\_OUTPUT\_1);

outputExpander.digitalWrite(LIGHT\_OUTPUT\_1, !currentState);

client.publish("home/lights/living/status", !currentState ? "ON" : "OFF");

}

prevSwitch1 = switch1;

}

delay(100);

}

**8.2 Industrial Monitoring**

**8.2.1 Production Line Monitoring System**

#include <WiFi.h>

#include <PubSubClient.h>

#include <Wire.h>

#include <Adafruit\_MCP23017.h>

#include <ModbusMaster.h>

// Network settings

const char\* ssid = "Factory\_Network";

const char\* password = "SecurePassword";

const char\* mqtt\_server = "192.168.10.50";

// Modbus settings

#define MAX485\_DE 27

#define MAX485\_RE 27

#define RS485\_SERIAL Serial2

ModbusMaster modbus;

Adafruit\_MCP23017 ioExpander;

// Sensor mappings

const uint8\_t EMERGENCY\_STOP = 0;

const uint8\_t MACHINE\_RUNNING = 1;

const uint8\_t FAULT\_INDICATOR = 2;

// Output mappings

const uint8\_t WARNING\_LIGHT = 0;

const uint8\_t ALARM\_BUZZER = 1;

// Modbus device addresses

const uint8\_t TEMPERATURE\_CONTROLLER = 1;

const uint8\_t FLOW\_METER = 2;

const uint8\_t PRESSURE\_SENSOR = 3;

// Analog input scaling

float scaleCurrentInput(uint16\_t raw) {

// Scale 4-20mA input to engineering units

// Example for temperature range 0-100°C

return (raw / 4095.0) \* 16.0 + 4.0; // Convert to mA

float percent = (mA - 4.0) / 16.0; // Convert to percentage

return percent \* 100.0; // Scale to temperature

}

void preTransmission() {

digitalWrite(MAX485\_DE, HIGH);

digitalWrite(MAX485\_RE, HIGH);

}

void postTransmission() {

digitalWrite(MAX485\_DE, LOW);

digitalWrite(MAX485\_RE, LOW);

}

void setup() {

Serial.begin(115200);

// Initialize I/O

Wire.begin();

ioExpander.begin(0x20);

// Configure digital I/O

for (uint8\_t i = 0; i < 8; i++) {

ioExpander.pinMode(i, INPUT);

ioExpander.pullUp(i, HIGH);

}

for (uint8\_t i = 0; i < 8; i++) {

ioExpander.pinMode(i + 8, OUTPUT);

ioExpander.digitalWrite(i + 8, LOW);

}

// Initialize RS485

pinMode(MAX485\_DE, OUTPUT);

pinMode(MAX485\_RE, OUTPUT);

digitalWrite(MAX485\_DE, LOW);

digitalWrite(MAX485\_RE, LOW);

RS485\_SERIAL.begin(9600, SERIAL\_8N1, 16, 17);

modbus.begin(1, RS485\_SERIAL);

modbus.preTransmission(preTransmission);

modbus.postTransmission(postTransmission);

// Connect to Wi-Fi

WiFi.begin(ssid, password);

// Connect to MQTT

client.setServer(mqtt\_server, 1883);

}

void loop() {

// Ensure MQTT connection

if (!client.connected()) {

reconnectMQTT();

}

client.loop();

// Check digital inputs

bool emergencyStop = !ioExpander.digitalRead(EMERGENCY\_STOP);

bool machineRunning = !ioExpander.digitalRead(MACHINE\_RUNNING);

bool faultCondition = !ioExpander.digitalRead(FAULT\_INDICATOR);

// Read temperature from Modbus device

uint8\_t result = modbus.readHoldingRegisters(0x00, 2);

float temperature = 0;

if (result == modbus.ku8MBSuccess) {

temperature = modbus.getResponseBuffer(0) / 10.0;

client.publish("factory/line1/temperature", String(temperature).c\_str());

}

// Read analog inputs

uint16\_t pressureRaw = analogRead(34); // 4-20mA input

float pressure = scaleCurrentInput(pressureRaw);

client.publish("factory/line1/pressure", String(pressure).c\_str());

// Process machine state

if (emergencyStop) {

ioExpander.digitalWrite(WARNING\_LIGHT, HIGH);

ioExpander.digitalWrite(ALARM\_BUZZER, HIGH);

client.publish("factory/line1/status", "EMERGENCY\_STOP");

} else if (faultCondition) {

ioExpander.digitalWrite(WARNING\_LIGHT, HIGH);

ioExpander.digitalWrite(ALARM\_BUZZER, machineRunning);

client.publish("factory/line1/status", "FAULT");

} else if (machineRunning) {

ioExpander.digitalWrite(WARNING\_LIGHT, LOW);

ioExpander.digitalWrite(ALARM\_BUZZER, LOW);

client.publish("factory/line1/status", "RUNNING");

} else {

ioExpander.digitalWrite(WARNING\_LIGHT, LOW);

ioExpander.digitalWrite(ALARM\_BUZZER, LOW);

client.publish("factory/line1/status", "STOPPED");

}

// Temperature alarm

if (temperature > 85.0) {

ioExpander.digitalWrite(WARNING\_LIGHT, HIGH);

client.publish("factory/line1/alarm", "TEMPERATURE\_HIGH");

}

delay(1000);

}

void reconnectMQTT() {

while (!client.connected()) {

if (client.connect("FactoryController")) {

client.subscribe("factory/line1/commands");

} else {

delay(5000);

}

}

}

**8.3 Remote Monitoring with GSM**

#include <TinyGSM.h>

#include <Wire.h>

#include <Adafruit\_MCP23017.h>

// Define modem type

#define TINY\_GSM\_MODEM\_SIM800

#define SerialAT Serial1

// APN settings

const char\* apn = "internet";

const char\* user = "";

const char\* pass = "";

// Server details

const char\* server = "example.com";

const int port = 80;

TinyGsm modem(SerialAT);

TinyGsmClient client(modem);

Adafruit\_MCP23017 ioExpander;

// Sensor pins

const int TEMP\_SENSOR = 34; // Analog input

const int LEVEL\_SENSOR = 35; // Analog input

const int ALARM\_INPUT = 0; // Digital input on MCP23017

// Variables for sensor readings

float temperature, level;

bool alarm;

// Last upload time

unsigned long lastUploadTime = 0;

const unsigned long uploadInterval = 900000; // 15 minutes

void setup() {

Serial.begin(115200);

SerialAT.begin(115200, SERIAL\_8N1, 26, 25); // RX=GPIO26, TX=GPIO25

// Initialize I/O

Wire.begin();

ioExpander.begin(0x21);

ioExpander.pinMode(ALARM\_INPUT, INPUT);

ioExpander.pullUp(ALARM\_INPUT, HIGH);

// Initialize modem

Serial.println("Initializing modem...");

modem.restart();

String modemInfo = modem.getModemInfo();

Serial.print("Modem: ");

Serial.println(modemInfo);

// Connect to mobile network

Serial.print("Waiting for network...");

if (!modem.waitForNetwork()) {

Serial.println(" fail");

return;

}

Serial.println(" OK");

Serial.print("Signal quality: ");

Serial.println(modem.getSignalQuality());

Serial.print("Connecting to ");

Serial.print(apn);

if (!modem.gprsConnect(apn, user, pass)) {

Serial.println(" fail");

return;

}

Serial.println(" OK");

}

void loop() {

// Read sensors

temperature = readTemperature();

level = readLevel();

alarm = !ioExpander.digitalRead(ALARM\_INPUT);

// Print data to serial

Serial.print("Temperature: ");

Serial.print(temperature);

Serial.println(" °C");

Serial.print("Level: ");

Serial.print(level);

Serial.println(" %");

Serial.print("Alarm: ");

Serial.println(alarm ? "YES" : "NO");

// Check if it's time to upload or if there's an alarm

if (millis() - lastUploadTime > uploadInterval || alarm) {

uploadData();

lastUploadTime = millis();

}

delay(60000); // Check every minute

}

float readTemperature() {

// Read 4-20mA temperature sensor

int rawValue = analogRead(TEMP\_SENSOR);

float mA = (rawValue / 4095.0) \* 16.0 + 4.0;

// Example conversion for a sensor with range -50 to 150°C

float temperature = ((mA - 4.0) / 16.0) \* 200.0 - 50.0;

return temperature;

}

float readLevel() {

// Read 4-20mA level sensor

int rawValue = analogRead(LEVEL\_SENSOR);

float mA = (rawValue / 4095.0) \* 16.0 + 4.0;

// Example conversion for a 0-100% level sensor

float level = ((mA - 4.0) / 16.0) \* 100.0;

return level;

}

void uploadData() {

Serial.println("Connecting to server...");

if (!client.connect(server, port)) {

Serial.println("Connection failed");

return;

}

Serial.println("Connected to server");

// Prepare the data in JSON format

String data = "{\"device\_id\":\"CL001\",\"temperature\":";

data += String(temperature);

data += ",\"level\":";

data += String(level);

data += ",\"alarm\":";

data += alarm ? "true" : "false";

data += "}";

// Prepare the HTTP POST request

String httpRequest = "POST /api/data HTTP/1.1\r\n";

httpRequest += "Host: ";

httpRequest += server;

httpRequest += "\r\n";

httpRequest += "Content-Type: application/json\r\n";

httpRequest += "Content-Length: ";

httpRequest += data.length();

httpRequest += "\r\n\r\n";

httpRequest += data;

// Send the request

client.print(httpRequest);

// Wait for the response

unsigned long timeout = millis();

while (client.connected() && millis() - timeout < 10000L) {

while (client.available()) {

char c = client.read();

Serial.print(c);

timeout = millis();

}

}

client.stop();

Serial.println("\nDisconnected from server");

}

**9. TROUBLESHOOTING**

**9.1 Common Issues and Solutions**

**9.1.1 Power Issues**

| **Issue** | **Possible Causes** | **Solutions** |
| --- | --- | --- |
| Board does not power up | • Incorrect power connection  • Insufficient power supply  • Blown fuse or protection circuit | • Check power supply polarity  • Verify power supply provides 9-12V DC with sufficient current  • Check voltage at power input terminals |
| Power cycles or resets | • Voltage drops under load  • Inadequate power supply  • Short circuit in attached devices | • Use higher current power supply  • Check for shorts in output connections  • Monitor voltage under load |

**9.1.2 Communication Issues**

| **Issue** | **Possible Causes** | **Solutions** |
| --- | --- | --- |
| Cannot connect via Wi-Fi | • Incorrect credentials  • Wi-Fi signal issues  • Network configuration problems | • Verify SSID and password  • Check Wi-Fi signal strength  • Try moving the device closer to router |
| No Ethernet connection | • Cable issues  • Network configuration  • W5500 module problems | • Try different Ethernet cable  • Verify network settings  • Check Ethernet LEDs for activity |
| RS485 communication fails | • Wiring issues  • Incorrect settings  • Termination problems | • Verify A/B terminal connections  • Check baud rate and format settings  • Add termination resistor if needed |
| GSM module not responding | • SIM card issues<br>• Signal problems<br>• APN configuration | • Verify SIM card is active  • Check signal strength  • Confirm APN settings |

**9.1.3 I/O Issues**

| **Issue** | **Possible Causes** | **Solutions** |
| --- | --- | --- |
| Digital inputs not working | • Incorrect wiring  • I2C address issues  • Software configuration | • Verify input connections  • Check MCP23017 address settings  • Confirm pull-up resistor configuration |
| MOSFET outputs not switching | • Incorrect wiring  • Load exceeds rating  • I2C communication issue | • Check output connections  • Verify load is within 500mA limit  • Test I2C communication with I2C scanner |
| Analog inputs reading incorrect values | • Sensor wiring issue  • Power supply problems  • Calibration needed | • Verify sensor connections  • Check sensor power supply  • Calibrate input scaling in software |
| Analog outputs not working | • Incorrect wiring  • I2C address issues  • Software configuration | • Check output connections  • Verify I2C communication with DAC  • Test with simple output program |

**9.2 Diagnostic Tools**

**9.2.1 I2C Scanner**

Use this code to identify I2C devices on the bus:

#include <Wire.h>

void setup() {

Serial.begin(115200);

Wire.begin();

Serial.println("I2C Scanner");

}

void loop() {

byte error, address;

int devices = 0;

Serial.println("Scanning...");

for (address = 1; address < 127; address++) {

Wire.beginTransmission(address);

error = Wire.endTransmission();

if (error == 0) {

Serial.print("I2C device found at address 0x");

if (address < 16)

Serial.print("0");

Serial.print(address, HEX);

Serial.println();

devices++;

}

}

if (devices == 0)

Serial.println("No I2C devices found");

delay(5000);

}

**9.2.2 GPIO Tester**

Use this code to test ESP32 GPIO pins:

// Change PIN\_TO\_TEST to test different pins

#define PIN\_TO\_TEST 2

void setup() {

Serial.begin(115200);

pinMode(PIN\_TO\_TEST, OUTPUT);

Serial.print("Testing GPIO ");

Serial.println(PIN\_TO\_TEST);

}

void loop() {

digitalWrite(PIN\_TO\_TEST, HIGH);

Serial.println("Pin HIGH");

delay(1000);

digitalWrite(PIN\_TO\_TEST, LOW);

Serial.println("Pin LOW");

delay(1000);

}

**9.2.3 Network Diagnostics**

#include <WiFi.h>

const char\* ssid = "YourNetworkName";

const char\* password = "YourPassword";

void setup() {

Serial.begin(115200);

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

int attempts = 0;

while (WiFi.status() != WL\_CONNECTED && attempts < 20) {

delay(500);

Serial.print(".");

attempts++;

}

if (WiFi.status() == WL\_CONNECTED) {

Serial.println("");

Serial.println("WiFi connected");

Serial.print("IP address: ");

Serial.println(WiFi.localIP());

// Test network connectivity

testConnectivity();

} else {

Serial.println("");

Serial.println("WiFi connection failed");

Serial.print("Status code: ");

Serial.println(WiFi.status());

}

}

void testConnectivity() {

Serial.println("Testing network connectivity...");

// Print connection details

Serial.print("SSID: ");

Serial.println(WiFi.SSID());

Serial.print("Signal strength (RSSI): ");

Serial.print(WiFi.RSSI());

Serial.println(" dBm");

Serial.print("MAC address: ");

Serial.println(WiFi.macAddress());

Serial.print("Subnet mask: ");

Serial.println(WiFi.subnetMask());

Serial.print("Gateway IP: ");

Serial.println(WiFi.gatewayIP());

Serial.print("DNS: ");

Serial.println(WiFi.dnsIP());

}

void loop() {

// Nothing in loop

}

**9.3 LED Indicators**

The board provides several LED indicators to help diagnose issues:

* **Power LED**: Indicates power supply is connected. If dim or off, check power supply.
* **Digital Input LEDs**: Illuminate when corresponding input is active. If not lighting when expected, check input wiring.
* **Output Status LEDs**: Indicate active outputs. If LED is on but load not working, check load wiring.
* **Communication LEDs**: Show activity on serial and network interfaces.

**9.4 Firmware Recovery**

If the board becomes unresponsive or the firmware is corrupted:

1. Connect the board via USB
2. Hold the BOOT button
3. Press and release the RESET button
4. Release the BOOT button
5. The board should now be in download mode
6. Use esptool.py to flash firmware:

esptool.py --chip esp32 --port COM3 --baud 921600 erase\_flash

esptool.py --chip esp32 --port COM3 --baud 921600 write\_flash -z 0x1000 firmware.bin

**10. TECHNICAL REFERENCE**

**10.1 ESP32 Pin Mapping**

| **ESP32 Pin** | **Function** | **Description** |
| --- | --- | --- |
| 3 | EN | Reset Button |
| 4 | GPIO36 (SENSOR\_VP) | 0-5V Analog Input Channel 1 |
| 5 | GPIO39 (SENSOR\_VN) | 0-5V Analog Input Channel 2 |
| 6 | GPIO34 | 4-20mA Analog Input Channel 1 |
| 7 | GPIO35 | 4-20mA Analog Input Channel 2 |
| 8 | GPIO32 | 433MHz RF Transmitter TX |
| 9 | GPIO33 | 433MHz RF Receiver RX |
| 10 | GPIO25 | GSM SIM800L/SIM7600E TX |
| 11 | GPIO26 | GSM SIM800L/SIM7600E RX |
| 12 | GPIO27 | MAX485 TXRX Control for MODBUS |
| 13 | GPIO14 | MCP23017 I2C INPUT Expander PORT A Interrupt |
| 14 | GPIO12 | Not Used |
| 16 | GPIO13 | MCP23017 I2C INPUT Expander PORT B Interrupt |
| 23 | GPIO15 | DHT22 Temperature/Humidity Sensor Channel 2 |
| 24 | GPIO2 | Buzzer (BEEP) |
| 25 | GPIO0 | BOOT Enable |
| 26 | GPIO4 | DHT22 Temperature/Humidity Sensor Channel 1 |
| 27 | GPIO16 | RS485 MODBUS MAX485 RO pin (RXD) |
| 28 | GPIO17 | RS485 MODBUS MAX485 DI pin (TXD) |
| 29 | GPIO5 | ETHERNET W5500 MODULE SPI Chip Select |
| 30 | GPIO18 | ETHERNET W5500 MODULE SPI SCLK |
| 31 | GPIO19 | ETHERNET W5500 MODULE SPI MISO |
| 33 | GPIO21 | I2C SDA |
| 34 | RXD0 | Debug/Programming (USB) RX |
| 35 | TXD0 | Debug/Programming (USB) TX |
| 36 | GPIO22 | I2C SCK |
| 37 | GPIO23 | ETHERNET W5500 MODULE SPI MOSI |

**10.2 MCP23017 Pin Configuration**

**10.2.1 Input Interface (U8)**

| **Pin** | **Name** | **Function** |
| --- | --- | --- |
| 1-8 | GPB0-GPB7 | Digital Inputs 1-8 (CN3) |
| 9 | VDD | 3.3V |
| 10 | VSS | GND |
| 11 | NC | Not Connected |
| 12 | SCK | I2C SCK |
| 13 | SDA | I2C SDA |
| 14 | NC | Not Connected |
| 15 | A0 | Address Line 0 |
| 16 | A1 | Address Line 1 |
| 17 | A2 | Address Line 2 |
| 18 | Reset | Reset |
| 19 | INTB | Connected to ESP32 GPIO13 |
| 20 | INTA | Connected to ESP32 GPIO14 |
| 21-28 | GPA0-GPA7 | Various Special Functions |

**10.2.2 Output Interface (U26)**

| **Pin** | **Name** | | **Function** |
| --- | --- | --- | --- |
| 1-4 | GPB0-GPB3 | MOSFET Outputs 9-12 (Q11-Q14) | |
| 5-8 | GPB4-GPB7 | Not Connected | |
| 9 | VDD | 3.3V | |
| 10 | VSS | GND | |
| 11 | NC | Not Connected | |
| 12 | SCK | I2C SCK | |
| 13 | SDA | I2C SDA | |
| 14 | NC | Not Connected | |
| 15 | A0 | Address Line 0 | |
| 16 | A1 | Address Line 1 | |
| 17 | A2 | Address Line 2 | |
| 18 | Reset | Reset | |
| 19 | INTB | Not Connected | |
| 20 | INTA | Not Connected | |
| 21-28 | GPA0-GPA7 | MOSFET Outputs 1-8 (Q3-Q10) | |

**10.3 GP8413 Analog Output Interface (U46)**

| **Pin** | **Name** | **Function** |
| --- | --- | --- |
| 1 | SCK | I2C SCK |
| 2 | SDA | I2C SDA |
| 3 | A0 | Address Line 0 |
| 4 | A1 | Address Line 1 |
| 5 | VCC | 12V Supply |
| 6 | GND | GND |
| 7 | VOUT1 | 0-5V / 0-10V Analog Output Channel 1 |
| 8 | VOUT2 | 0-5V / 0-10V Analog Output Channel 2 |
| 9 | A4 | Address Line 2 |
| 10 | VSS | GND |

**10.4 I2C Address Configuration**

| **Device** | **Function** | **Default Address** | **Address Selection** |
| --- | --- | --- | --- |
| MCP23017 (U8) | Digital Inputs | 0x21 | JP1 short (0x21) |
| MCP23017 (U26) | MOSFET Outputs | 0x20 | Open jumpers (0x20) |
| GP8413 (U46) | Analog Outputs | 0x58 | Default 0x58 |

**10.5 Technical Specifications**

**10.5.1 Electrical Specifications**

| **Parameter** | **Specification** |
| --- | --- |
| Input Voltage | 9-12V DC (24V maximum) |
| Power Consumption | 1W (idle), 10W (full load) |
| Digital Input Type | Optically isolated, dry contact |
| Digital Output Type | N-channel MOSFET, low-side switching |
| Output Rating | 12/24V DC, 500mA per channel |
| Analog Input Range | 4-20mA (CH1-2), 0-5V DC (CH3-4) |
| Analog Output Range | 0-10V DC |
| RS485 Interface | Half-duplex, Modbus RTU compatible |

**10.5.2 Environmental Specifications**

| **Parameter** | **Specification** |
| --- | --- |
| Operating Temperature | -40°C to +85°C |
| Storage Temperature | -40°C to +100°C |
| Humidity | 10% to 90% RH (non-condensing) |
| Altitude | Up to 2000m |
| Protection Rating | IP00 (requires enclosure) |

**10.5.3 Mechanical Specifications**

| **Parameter** | **Specification** |
| --- | --- |
| Dimensions | 200mm × 110mm × 45mm (L × W × H) |
| Weight | Approximately 250g |
| Mounting | 4× mounting holes (4mm diameter) |
| Terminal Connectors | Spring or screw terminals |

**11. APPENDICES**

**11.1 Certifications**

* **CE Certified**: Compliant with European safety standards
* **RoHS Compliant**: Free from hazardous substances
* **FCC Compliant**: For RF communications

**11.2 Warranty Information**

The Cortex Link A8F-M ESP32 includes a standard 12-month warranty against manufacturing defects. Extended warranty options are available upon request.

**11.3 Product Customization**

The Cortex Link A8F-M ESP32 can be customized to meet specific requirements. Contact MESA directly for custom configurations, including:

* Modified I/O configurations
* Custom communication interfaces
* Specialized firmware
* Alternate enclosure options
* OEM branding

**11.4 Technical Support**

For technical support, contact MESA:

* Email: support@mesa-automation.com
* Website: www.mesa-automation.com
* Phone: [Contact Information]

**11.5 Additional Resources**

* Sample code repository: [GitHub Link]
* Application notes: [Website Link]
* Video tutorials: [YouTube Channel]
* User forum: [Forum Link]

© 2025 Microcode Embedded Systems and Automation (MESA)  
All Rights Reserved

*Document Revision: 1.1*  
*Last Updated: April 28, 2025*