ESP32-C3 Modular IoT System

Technical Documentation

TI0162 - Internet of Things Project

Marcelo Correa

mvcorrea@gmail.com

September 15, 2025

Abstract

This document presents the technical documentation for a complete IoT system developed for ESP32-C3 microcontroller using Rust programming language and the Embassy async framework. The system implements a modular architecture for environmental data collection via BME280 sensor, WiFi connectivity, MQTT communication, and interactive serial console interface. The project demonstrates embedded systems development best practices with asynchronous programming, real-time data processing, and industrial IoT communication protocols. Future extensions include PID-controlled PWM devices for environmental control applications.

Contents

1 Introduction

1.1 Project Overview

The ESP32-C3 Modular IoT System is a comprehensive embedded solution designed for environmental monitoring and control applications. Built using Rust programming language and the Embassy async framework, the system provides a robust, scalable platform for IoT deployments.

1.2 Key Features

- **Modular Architecture**: Independent modules for sensor, connectivity, and communication
- **Asynchronous Processing**: Embassy framework for non-blocking operations
- **Real-time Communication**: MQTT protocol for industrial IoT integration
- **Interactive Console**: USB Serial/JTAG interface for configuration and monitoring
- **Environmental Sensing**: BME280 sensor for temperature, humidity, and pressure
- **Wireless Connectivity**: WiFi with automatic reconnection and DHCP

1.3 Target Applications

- Environmental monitoring systems
- Smart building automation
- Industrial process monitoring
- Agricultural IoT solutions
- Research and educational platforms

2 System Architecture

2.1 Hardware Platform

The system is built around the ESP32-C3 microcontroller, specifically targeting the WeAct ESP32-C3 development board. The ESP32-C3 features:

- RISC-V single-core processor at 160 MHz
- 400 KB SRAM, 384 KB ROM
- WiFi 802.11 b/g/n support
- Built-in USB Serial/JTAG controller
- 22 programmable GPIOs
- Multiple communication interfaces (I2C, SPI, UART)

2.2 Software Architecture

The system follows a modular architecture with clear separation of concerns. Each module is independently developed, tested, and maintained, allowing for easy extension and modification.

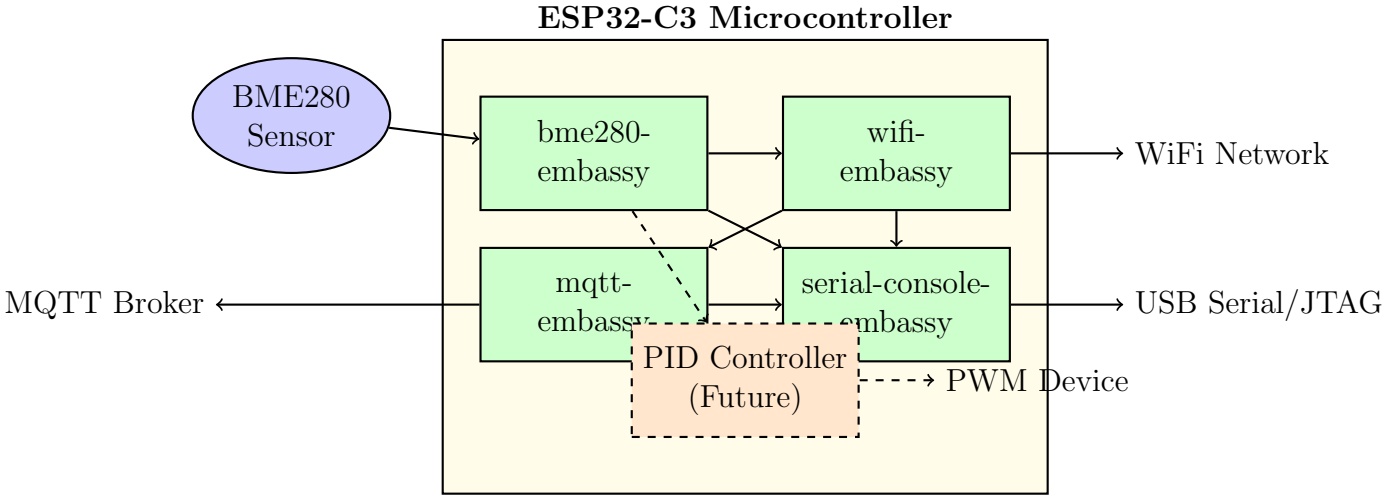

Figure 1: System Architecture Overview

2.3 Module Description

2.3.1 bme280-embassy Module

Handles asynchronous communication with the BME280 environmental sensor via I2C interface. Provides:

- Temperature reading (-40°C to +85°C)
- Humidity measurement (0-100% RH)
- Atmospheric pressure (300-1100 hPa)
- Automatic calibration coefficient application
- Real-time data acquisition at configurable intervals

2.3.2 wifi-embassy Module

Manages WiFi connectivity with robust error handling and automatic reconnection:

- WPA2/WPA3 security support
- DHCP client for automatic IP configuration
- Connection monitoring and automatic recovery
- Network stack integration for TCP/UDP operations
- Signal strength monitoring

2.3.3 mqtt-embassy Module

Implements MQTT 3.1.1 client for industrial IoT communication:

- QoS 0 message publishing
- JSON-formatted sensor data transmission
- Configurable broker connection parameters
- Automatic reconnection on network failures
- Heartbeat and system status reporting

2.3.4 serial-console-embassy Module

Provides interactive console interface via USB Serial/JTAG:

- Real-time system configuration
- WiFi credential management
- MQTT broker configuration
- System status monitoring
- Command-line interface with help system

3 Hardware Configuration

3.1 Pin Assignment

The system uses the following GPIO pin configuration:

Function	GPIO Pin	Description
I2C SDA	GPIO8	BME280 data line
I2C SCL	GPIO9	BME280 clock line
Status LED	GPIO3	System status indicator
UART TX	GPIO21	Serial console (alternative)
UART RX	GPIO20	Serial console (alternative)
USB D+	Built-in	USB Serial/JTAG data+
USB D-	Built-in	USB Serial/JTAG data-

Table 1: GPIO Pin Assignment

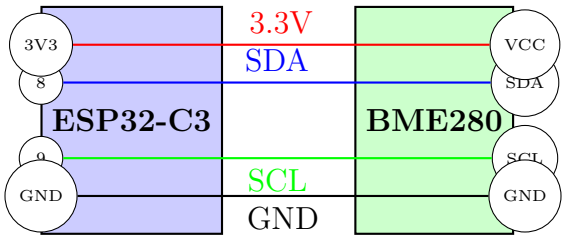

Figure 2: BME280 Sensor Connection Diagram

3.2 BME280 Connection

4 Software Implementation

4.1 Development Environment

The project is developed using:

- **Rust 2021 Edition**: Modern systems programming language
- **Embassy Framework**: Async runtime for embedded systems
- **esp-hal**: Hardware Abstraction Layer for ESP32-C3
- **probe-rs**: Debugging and flashing tool
- **RTT (Real-Time Transfer)**: Debug output mechanism

4.2 Key Dependencies

Listing 1: Core Dependencies (Cargo.toml)

```
[dependencies]
# ESP32-C3 Hardware Abstraction Layer
esp-hal = { version = "1.0.0-rc.0", features = ["esp32c3", "unstable"] }
esp-hal-embassy = { version = "0.9.0", features = ["esp32c3"] }

# Embassy Async Framework
embassy-executor = { version = "0.7", features = ["task-arena-size
    -32768"] }
embassy-time = { version = "0.4" }
embassy-sync = { version = "0.7" }

# Communication and I/O
embedded-io-async = "0.6"
heapless = "0.8"

# Debugging
rtt-target = "0.5"
panic-rtt-target = "0.1"
```

4.3 Module Integration Pattern

Each module follows a consistent pattern for integration:

Listing 2: Module Integration Example

```
use embassy_executor::Spawner;
use embassy_time::{Duration, Timer};

#[embassy_executor::task]
async fn sensor_task() {
    let mut bme280 = BME280::new().await;

    loop {
        match bme280.read_all().await {
            Ok(data) => {
                rprintln!("Temperature: {:.2} C ", data.temperature);
                rprintln!("Humidity: {:.2}%", data.humidity);
                rprintln!("Pressure: {:.2} hPa", data.pressure);
            }
            Err(e) => rprintln!("Sensor error: {:?}", e),
        }

        Timer::after(Duration::from_secs(30)).await;
    }
}

#[esp_hal::main]
async fn main(spawner: Spawner) {
    // Initialize hardware and spawn tasks
    spawner.spawn(sensor_task()).ok();

    // Main loop
    loop {
        Timer::after(Duration::from_secs(1)).await;
    }
}
```

5 Data Flow and Communication

5.1 MQTT Message Format

The system publishes sensor data in JSON format:

Listing 3: Sensor Data Message

```
{
    "timestamp": "2025-01-15T10:30:00Z",
    "sensor": "BME280",
    "data": {
        "temperature": 23.5,
        "humidity": 65.2,
        "pressure": 1013.25
    },
    "metadata": {
        "device_id": "esp32-c3-001",
        "firmware_version": "1.0.0",
```

```
        "uptime": 3600
    }
}
```

5.2 System Status Messages

Listing 4: System Status Message

```
{
    "status": "online",
    "uptime": 7200,
    "free_heap": 45000,
    "wifi_rssi": -42,
    "sensor_status": "active",
    "mqtt_status": "connected"
}
```

6 Future Enhancements

6.1 PID Control System

The architecture is designed to support future integration of PID (Proportional-Integral-Derivative) controllers for environmental control applications.

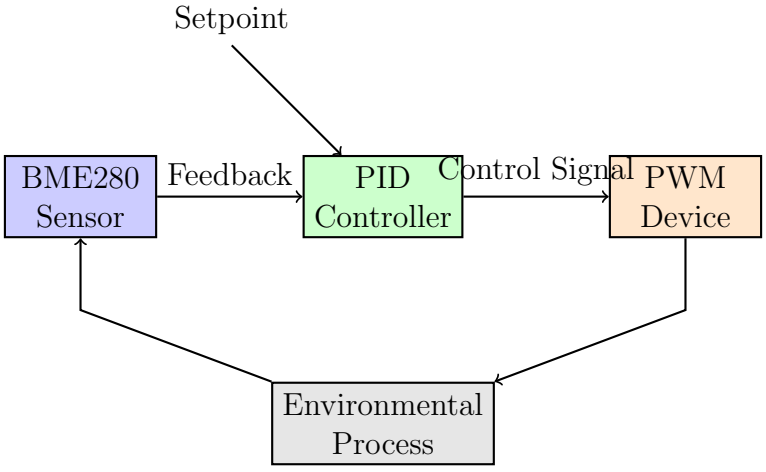

Figure 3: Planned PID Control Loop Integration

The PID controller will implement the standard control algorithm:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{1}$$

Where:

- $u(t)$ is the control output
- $e(t)$ is the error signal (setpoint - measured value)
- K_p, K_i, K_d are the proportional, integral, and derivative gains

6.2 Planned Features

- **Environmental Control**: Temperature, humidity regulation
- **PWM Output**: Variable speed fans, heaters, actuators
- **Multi-zone Control**: Independent control of multiple zones
- **Web Interface**: Browser-based monitoring and configuration
- **Data Logging**: Historical data storage and analysis
- **Alarm System**: Threshold-based alerts and notifications

7 Performance and Specifications

7.1 System Performance

Parameter	Value
Sensor Reading Interval	30 seconds
MQTT Publish Rate	30 seconds
WiFi Reconnection Time	¡ 10 seconds
Console Response Time	¡ 100 ms
Memory Usage	¡ 200 KB
Power Consumption	¡ 100 mA @ 3.3V

Table 2: System Performance Specifications

7.2 Sensor Specifications

Parameter	Range	Accuracy
Temperature	-40°C to +85°C	±0.5°C
Humidity	0 to 100% RH	±3% RH
Pressure	300 to 1100 hPa	±1 hPa

Table 3: BME280 Sensor Specifications

8 Conclusion

The ESP32-C3 Modular IoT System represents a comprehensive solution for environmental monitoring and control applications. The modular architecture, combined with modern Rust programming practices and the Embassy async framework, provides a robust foundation for industrial IoT deployments.

The system's design emphasizes:

- **Modularity**: Easy to extend and modify
- **Reliability**: Robust error handling and recovery
- **Performance**: Efficient async processing
- **Maintainability**: Clear code structure and documentation
- **Scalability**: Designed for future enhancements

Future developments will focus on expanding the control capabilities with PID algorithms and additional sensor integration, making this platform suitable for a wide range of industrial and commercial applications.

9 References

1. ESP32-C3 Technical Reference Manual, Espressif Systems
2. BME280 Environmental Sensor Datasheet, Bosch Sensortec
3. Embassy Framework Documentation, https://embassy.dev
4. MQTT Version 3.1.1 Specification, OASIS Standard
5. Rust Programming Language Documentation, https://doc.rust-lang.org
6. PID Control Theory, https://en.wikipedia.org/wiki/Proportional-integral-derivative_controller

A Build Instructions

A.1 Prerequisites

Listing 5: Development Environment Setup

```
# Install Rust
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

# Add ESP32-C3 target
rustup target add riscv32imc-unknown-none-elf

# Install probe-rs
cargo install probe-rs --features cli

# Clone repository
git clone <repository-url>
cd workspace
```

A.2 Building and Flashing

Listing 6: Build and Flash Commands

```
# Build individual modules
cd bme280-embassy
cargo build --release

cd ../wifi-embassy
cargo build --release

cd ../mqtt-embassy
cargo build --release

cd ../serial-console-embassy
cargo run --example direct_usb_console --release
```

B Configuration Files

B.1 Environment Configuration

Listing 7: .cargo/config.toml

```
[env]
WIFI_SSID = "YourNetworkName"
WIFI_PASSWORD = "YourPassword"
MQTT_BROKER_IP = "192.168.1.100"
MQTT_BROKER_PORT = "1883"
MQTT_CLIENT_ID = "esp32-c3-device"
MQTT_TOPIC_PREFIX = "esp32"
```