# WT32-ETH01

# Getting Started Guide for AWS IoT Core

## Table of Contents



## 1 Document information

### 1.1 Document revision history

| Date | Modified by | Description |
|---|---|---|
| October 21, 2024 | Vans | First release |

### 1.2 Applicable operating systems for this guide

This guide is applicable to the ESP32 series chips with FreeRTOS system. It adopts ESP-IDF, an Internet of Things operating system independently developed by Espressif.

## 2 Overview

WT32-ETH01 is an embedded serial port to Ethernet module based on the ESP32 series launched by Wireless-tag Technology Co., Ltd. The module internally integrates an optimized TCP/IP protocol stack, which makes it easy for users to complete the networking function of embedded devices and greatly reduces the development time cost. The WT32-ETH01 can easily and quickly connect to AWS IoT Core.

Moreover, the module is compatible with half pad and connector through-hole designs. The board width is a universal width. The module can be directly soldered on the board, or a

connector can be soldered, or it can also be used on a breadboard, making it convenient for users to use in different scenarios.

The ESP32 series IC is a SoC that integrates 2.4GHz Wi-Fi and Bluetooth dual-mode. It has ultra-high radio frequency performance, stability, versatility and reliability, as well as ultra-low power consumption.

# 3 Hardware description

## 3.1 Datasheet

The link to the product datasheet: *https://en.wireless-tag.com/product-item-2.html*.

## 3.2 Standard kit contents

The contents of the standard shipping hardware package as indicated below:

- ESP32 module: WT32-S1, with 32Mbit flash onboard
- A highly integrated Ethernet transceiver：LAN8720A
- Network port: RJ45, 10/100Mbps
- Debug port: UART
- Power supply: 5V or 3.3V TTL
- Packing: Half pad/ Through-hole connector (Optional)

Please links to the page on our company website for more detail:
*https://en.wireless-tag.com/product-item-2.html.*

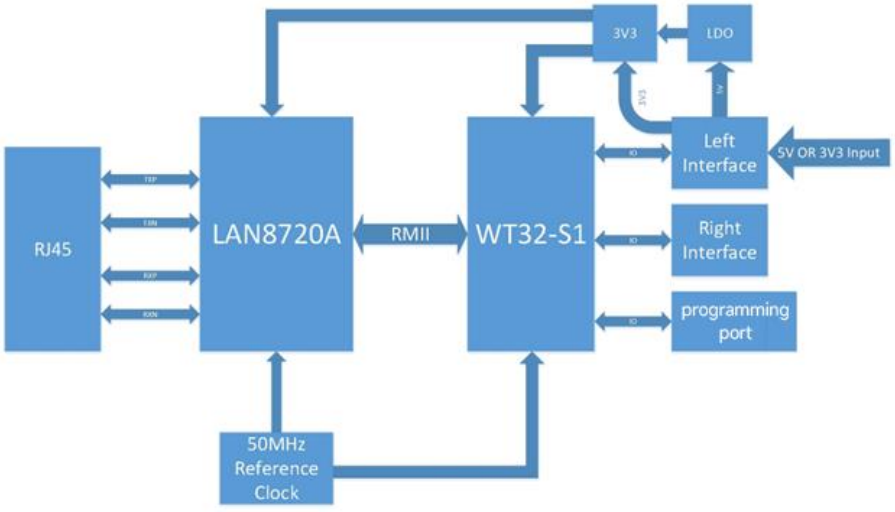
## 3.3 User provided items

A USB-to-TTL serial debugging tool is needed for development.

## 3.4 3rd party purchasable items

None.

## 3.5 Additional hardware references



# 4 Set up your development environment

## 4.1 Tools installation (IDEs, Toolchains, SDKs)

.

1. IDE based
    - *Eclipse Plugin*
    - *VSCode Extension*
    - *Arduino*

2. idf.py

The idf.py command-line tool provides a front-end for easily managing your project builds, deployment and debugging, and more. It manages several tools, for example:

- CMake, which configures the project to be built.
- Ninja, which builds the project.
- Esptool.py, which flashes the target.

Can read more about configuring the build system using idf.py here.

3. ESP-IDF

For the manual procedure, please select according to your operating system.

- Windows Installer
- Linux and macOS

4. Optimizing the compiler

In ESP-IDF, you can force compiler optimization by modifying the compiler options. You can set the compiler optimization options in the CMakeLists.txt file.
In your project's CMakeLists.txt file, add the following lines to set the compiler optimization options

```
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -O3")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O3")
```

5. SDK

ESP-IDF , A tutorial on setting up the sdk environment can be found in the Getting Started guide.
ESP-IDF is Espressif's official IoT Development Framework for the ESP32, ESP32-S, ESP32-C and ESP32-H series of SoCs.
It provides a self-sufficient SDK for any generic application development on these platforms, using programming languages such as C and C++.

## 4.2 Additional software references

ESP32 Chip Manual - Provides specifications for the ESP32 chip
ESP-IDF Programming Guide for ESP32 – Extensive documentation for the ESP-IDF evelopment framework

# 5 Set up device hardware

## 5.1 Product Images


## 5.2 Pin Description

| Pin | Name | Description |
|---|---|---|
| 1 | EN | Reserved debugging/burning interface; Active-high enable |
| 2 | GND | Reserved debugging/burning interface; GND |
| 3 | 3V3 | Reserved debugging/burning interface; 3V3 |
| 4 | TXD | Reserved debugging/burning interface; IO1, TXD0 |
| 5 | RXD | Reserved debugging/burning interface; IO3, RXD0 |
| 6 | IO0 | Reserved debugging/burning interface; IO0 |

**Table 1 Debugging/Burning Interfaces**

| Pin | Name | Description |
|---|---|---|
| 1 | EN | Active-high enable |
| 2 | CFG | Reserved debugging/burning interface; GND |
| 3 | 485_EN | IO17, TXD2 |
| 4 | TXD | IO33, RS485 Enable pin |
| 5 | RXD | IO5, RXD2 |
| 6 | GND | GND |
| 7 | 3V3 | 3V3 Power supply |
| 8 | GND | GND |
| 9 | 5V | 5V Power supply |
| 10 | LINK | Network connection indicator pin |
| 11 | GND | GND |
| 12 | IO39 | IO39, Input only |
| 13 | IO36 | IO36, Input only |
| 14 | IO15 | IO15 |
| 15 | IO14 | IO14 |
| 16 | IO12 | IO12 |
| 17 | IO35 | IO35, Input only |
| 18 | IO4 | IO4 |
| 19 | IO2 | IO2 |
| 20 | GND | GND |

**Table 2 : IO Descriptions**

Note 1: The module enables high level by default.
Note 2: Power supply makes a binary choice between 3V3 and 5V.
Note 3: IO39, IO35 and IO36 only support input.

## 5.3 Power Supply Characteristics

1. Supply Voltage

You can make a binary choice between 3V3 and 5V for power supply voltage of the module.

2. Power Supply Modes

Users can choose from the following modes flexibly according to their needs:

1) Through hole (Welding pins):
   - Power supply with Dupont line connection;
   - Power supply with breadboard connection;

2) Half pad (Directly welded to the board): Power supply of user board.

## 5.4 Boot Configurations

The chip can be configured with the following startup parameters via the Strapping pin at power-up or hardware reset.

Strapping pin: GPIO0 and GPIO2

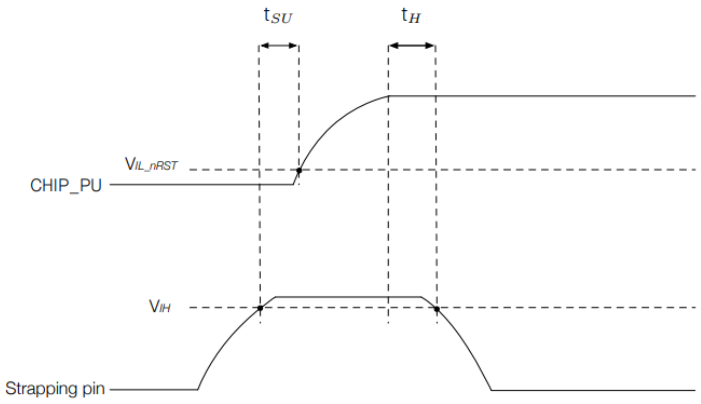
| Strapping Pin | Default Configuration | Bit Value |
|---|---|---|
| GPIO0 | Pull-up | 1 |
| GPIO2 | Pull-down | 0 |

**Default Configuration of Strapping Pins**

The timing of signals connected to the strapping pins should adhere to the setup time and hold time specifications in Table and Figure.

| Parameter | Description | Min (ms) |
|---|---|---|
| $t_{SU}$ | Setup time is the time reserved for the power rails to stabilize before the CHIP_PU pin is pulled high to activate the chip | 0 |
| $t_H$ | Hold time is the time reserved for the chip to read the strapping pin values after CHIP_PU is already high and before these pins start operating as regular IO pins. | 3 |

**Description of Timing Parameters for the Strapping Pins**




**Visualization of Timing Parameters for the Strapping Pins**

| Boot Mode | GPIO0 | GPIO2 |
|---|---|---|
| **SPI Boot Mode** | **1** | **Any value** |
| Joint Download Boot Mode | 0 | 0 |

**Chip Boot Mode Control**

Bold marks the default value and configuration.

## 5.5 LED Instructions

- LED1: Power light, when the power supply is normal, the indicator light is on;
- LED3: Serial port indicator, RXD2(IO5) When there is data flow, the indicator light is on;
- LED4: Serial port indicator, TXD2(IO17) When there is data flow, the indicator light is on;

# 6 Setup your AWS account and permissions

If you do not have an existing AWS account and user, refer to the online AWS documentation at Set up your AWS Account. To get started, follow the steps outlined in the sections below:

- Sign up for an AWS account
- Create an administrative user
- Open the AWS IoT console

Pay special attention to the Notes.

# 7 Create resources in AWS IoT

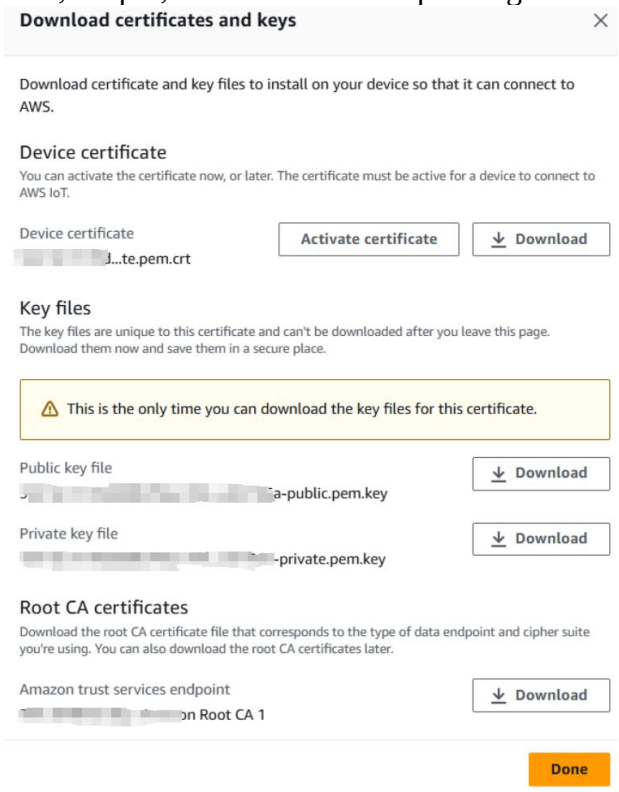
Refer to the online AWS documentation at Create AWS IoT Resources. Follow the steps outlined in these sections to provision resources for your device:

- Create an AWS IoT Policy
- Create a thing object

Pay special attention to the Notes.

# 8 Provision the device with credentials

During “Create a thing object”, you will encounter the requirement to download the certificate in the last step, as shown below, keep it, which is the corresponding server certificate of the device.



# 9 Build the demo

This tutorial uses the esp-idf/example/protocol/mqtt/ssl_mutual_auth example to test the device's connection to AWS-IoT-Core.

## 9.1 Engineering Configuration

1. After entering the project, you need to replace the three certificates in the main directory. The certificates to be replaced are stored in the connection toolkit you downloaded earlier. The replacement corresponds to the following:
   - The .client.crt file is the client certificate; use the .pem.crt file instead.
   - The .client.key file is the client key; use the .private.pem.key file instead.
   - The .mosquitto.org.crt file is the server-side secret key; use the CA1.pem file instead.
2. Replace the link to the mqtt server accessed by the project and add the client_id configuration entry.The link is replaced with the link used when connecting to the device, and the client_id used "basicPubSub".
   **Note: The link needs to be prefixed with mqtt://**
3. Activate the IDF environment, configure the chip as ESP32 and modify the WiFi configuration information of the project via menuconfig.
   **Note: Configuration path for WiFi configuration: Connection Configuration Example ->WiFi SSID / WiFi Password**

# 10 Run the demo

Take the project introduced in the previous chapter, burn it into your device, and open the serial port debugging assistant.

# 11 Verify messages in AWS IoT Core

Open "MQTT test cilent", set the Subscribe topic to "sdk/test/python" and click "Subscribe". button.

# 12 Troubleshooting

If the connection fails when connecting to mqtt, it is recommended to check whether the above steps have been completed, whether the client_id and uri have been correctly modified, and whether the certificate has been correctly replaced.
If there is something you don't understand, you can also refer to the AWS online documentation on *Troubleshooting AWS IoT*.