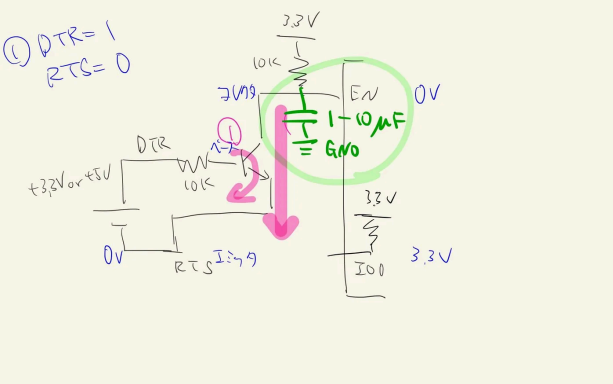
Open in app ↗ Sign up Sign in

Medium Search

[ESP32] Understanding the automatic boot loader mechanism and truth table

Kazutaka Yoshinaga · Follow
5 min read · Aug 2, 2024

Listen Share

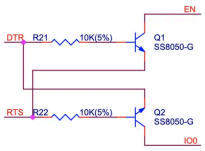


Currently developing "Mia," a talking cat-shaped robot that speaks dialect.

Mia
The talking cat-shaped robot Mia shares sadness and joy with more than 100 different rich expressions. It also speaks...
mia-cat.com

In order to create Mia's custom board (ESP32-based), I thought I had to first understand the schematic of the ESP32 development board, and when I started looking at it, I came across the following schematic and truth table and thought, "What the heck.

I started to investigate the following circuit as it corresponds to the automatic bootloader function of ESP32, but it took me 6 or 7 hours to understand it because I had to go back to the basics such as transistors and pull-up resistors after studying schematics for the first time in a long time.



Auto program

DTR	RTS	-->EN	IO0
1	1	1	1
0	0	1	1
1	0	0	1
0	1	1	0

In this issue, I would like to describe my understanding of how the ESP32 automatic upload function works. If you find any mistakes, we would appreciate it if you could point them out to us in the comments section.

UART flow control

First, UART stands for Universal Asynchronous Receiver/Transmitter.

There are two methods of transmitting digital signals: parallel and serial communication, and within serial communication there are synchronous and asynchronous types. UART falls into the asynchronous type of serial communication.

When exchanging data between two devices via UART, if data is sent from the other device when the device is not ready to receive the data, it will miss receiving the data. To prevent this from happening, the sending device is notified to "wait a moment before sending data," and the data **flow is adjusted** by temporarily suspending transmission or slowing down the speed. There are several methods of flow control, but the one that concerns us this time is hardware flow control.

- **Sender**: When the sending device wants to send data, it sets the RTS line High
- **Receiver**: When the RTS line goes High, the receiving device checks if it is ready to receive data from the sending device. If ready, the CTS (Clear to Send) line goes High to notify the sender of permission to send.

In this case, we assume that the program is written from the PC (sender side) to the ESP32 (receiver side) via USB.

1. The PC (sending side) prepares the program data to be written to the ESP32, and when it is ready for communication, it sets DTR High to notify the ESP32 that "communication is ready".
2. When the ESP32 (receiver) detects the DTR, it knows that the sender is ready to accept communication.
3. After communication is established, when the PC wants to send program data, it sets RTS High to signal "ready to send".
4. When ESP32 detects RTS, if the buffer is free, it raises CTS to High and notifies PC of "ready to receive".
5. The PC starts transmitting program data when CTS detects High.
6. When the ESP32 buffer is full, CTS is set Low to notify "reception pause".
7. When the PC detects a Low on CTS, it suspends data transmission.
8. When the ESP32 buffer is free, CTS is again set High to notify "resumption of reception".
9. The PC resumes data transmission, and when all data has been sent, DTR is set Low to signal "end of communication".

First, the **DTR tells the receiving terminal that communication has been established, and then the RTS asks the receiving terminal if it is OK to send data.** Without establishing communication first, data cannot be sent in the first place, and even if communication is established, if the receiving terminal does not have the capacity to accept the data, it will be overflowed, so the two-step process is to ask if it is OK to send the data now.

EN pin and IO0 pin of ESP32

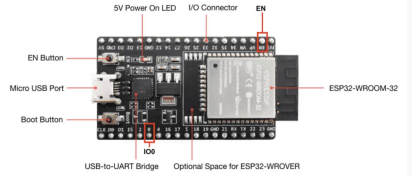
EN pin: Button for restart, abbreviation of Enable

- High: Normal operating mode. When the EN pin goes High, the ESP32 recovers from reset and starts normal operation.
- Low: The ESP32 enters a reset state. When the EN pin goes Low, ESP32 experiences a hard reset and all operations stop.
- Press when the program is re-run or behavior becomes unstable.
- There is no built-in pull-up resistor.

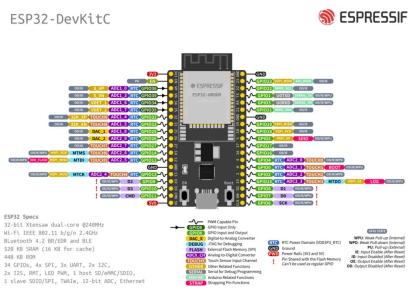
IO0 (Input/Output 0) pin

- High: In normal execution mode, the program stored in flash memory is executed.
- Low: Boot loader mode (also called flash mode), enters a special mode to write a new program to flash memory; ESP32 is ready to accept program writes.
- It has a **pull-up resistor built in,** so it is **normally High.**

So, when you want to write a new program to ESP32, you can do so by pressing the switch buttons in the following order.

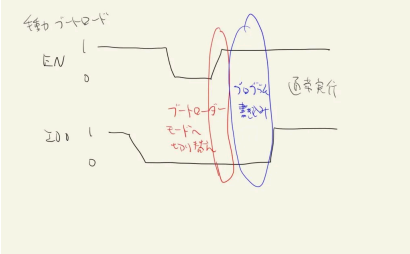


ESP32-DevKitC V4 with ESP32-WROOM-32 module soldered

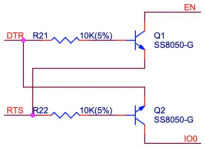


The above is just an example. To be precise, if IO0=EN=Low is maintained and only EN switches to High, that moment is detected as the boot loader mode and programming can start. It is fine as long as IO0=EN=Low state is maintained.

The above is just an example. To be precise, if IO0=EN=Low is maintained and only EN switches to High, that moment is detected as the boot loader mode and programming can start. It is fine as long as IO0=EN=Low state is maintained.



Automatic Boot Loader and Truth Table
Now, finally, we would like to look at the schematic and truth table here.



Auto program			
DTR	RTS-->EN	IO0	
1	1	1	1
0	0	1	1
1	0	0	1
0	1	1	0

First, 0 and 1 in this truth table represent the following, respectively.

1=High=Positive voltage (+5V or +3.3V)
0=Low=0V

This figure and the truth table show that by using the host PC program to adjust the voltages on the DTR and RTS pins of the USB-UART conversion chip, respectively, to create the state (DTR, RTS)=(1,1)(0,0)(1,0)(0,1), and also by using the two transistors (Q1, Q2) well, This shows that the EN and IO0 voltages can be well adjusted.

Role of Transistors
The transistor used here is an NPN transistor.
For more information on the role of transistors, please refer to the following article.

Read More at...

[ESP32] Understanding the automatic boot loader mechanism and truth table
The automatic bootloader of the ESP32 is a function that enables automatic programme writing when the PC instructs the...
kazulog.fun

Esp32 Bootloader Uart USB

Written by Kazutaka Yoshinaga

55 Followers · 59 Following

M.D., Serial entrepreneur, Engineer. He founded Melp, a web-based medical questionnaire, and exited to JMDC. Released Mia, a talking robot that speaks dialects

Follow

No responses yet

What are your thoughts?

Respond

More from Kazutaka Yoshinaga

[Protobuf] How Varint encoding works and the importance of field numbers

protobuf
Protocol Buffers

Kazutaka Yoshinaga

[Protobuf] How Varint encoding works and the importance of field numbers
What is a protocol buffer? Protobuf serializes data more efficiently than XML or JSON, with smaller message sizes and faster parsing.

Jun 15, 2024 1