ESP32 CAM

Starter Guide

Siegfried Kienzle

Contents

List of Figures

List of Tables

1 Hardware

This section describes which kind of hardware is required and should give an overview about the used boards.

1.1 ESP32-CAM Board

The ESP32-S-CAM board has different pins and connectors. Additional you will find a connector panel for the cam, a micro-sd-card-slot, two LED's, a reset button and a connector for an external antenna (this can be not used, because there is a resistance which uses the onboard pcb antenna [7]). The table 1 show you the board specifications.

CPU:		
	Processor:	Xtensa 32-bit LX6 [5]
	Cores:	2
	Clock frequency:	240 MHz [5]
	Performance:	600 DMIPS [5]
Memory:		
	ROM:	448 KiB [5]
	SRAM:	520 KiB [5]
	PSRAM:	4 MiB [5]
Wireless Connectivity:		
	Wi-Fi:	802.11 b/g/n [5]
	Bluetooth:	v4.2 BR/EDR and BLE Standards [5]
Wi-Fi Security:		WPA/WPA2/WPA2-Enterprise/WPS [6]
UART Baudrate (default):		115200 bps [6]

Table 1: Board specifications

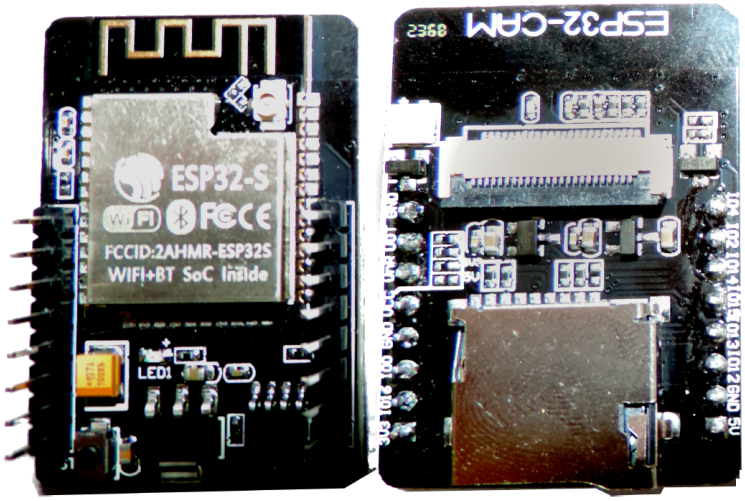

Figure 1: ESP32-S CAMSource: own picture

As shown in figure 2, you see the different pinouts of the ESP32-CAM board.

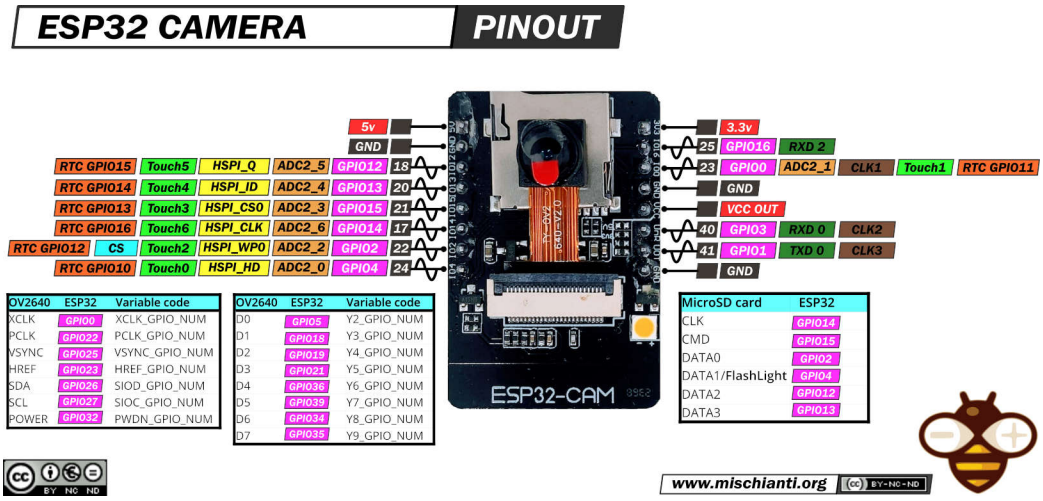

Figure 2: ESP32 CAMERA PINOUT
Source: https://www.mischianti.org/wp-content/uploads/2020/09/ESP32-CAM-pinout-mischianti.jpg
from Renzo Mischianti
License: https://creativecommons.org/licenses/by-nc-nd/4.0/

In the picture 3 you can see the backside of the ESP32-S CAM board. There are the reset-button (in the green box), the internal led (in the blue box) and the (not usable) connector for an external antenna (in the orange box).

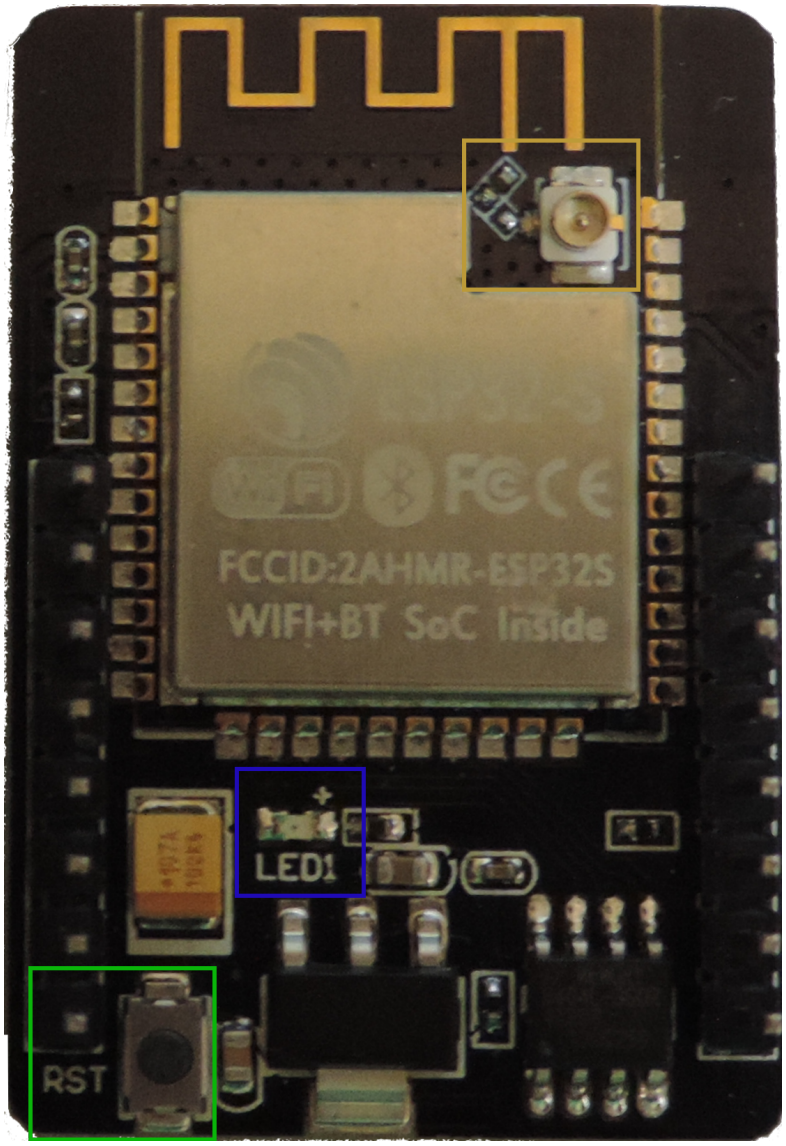

Figure 3: ESP32 CAMERA BacksideSource: own picture

1.2 ESP32-CAM MB Programmer Board

The ESP32-CAM MB Programmer is an optional board. It is a nice to have and with this board it is also possible (as the name already says) to program the ESP32-CAM. In this documentation it will not described how you can programm your ES32-CAM board with the MB programmer board. Additionally the most documentations describes how to upload source-code with the Arduino IDE, which will also not shown in this documentation.

Figure 4: ESP32-S CAM MB Programmer BoardSource: own picture

This board will be used later as power-supply for the ESP32-CAM board.

1.3 USB to TTL Serial Adapter

The USB to TTL serial adapter will be used for programming the ESP32-CAM Board. It is important that it supports 5V, because the ESP32-CAM board needs 5 V input voltage.

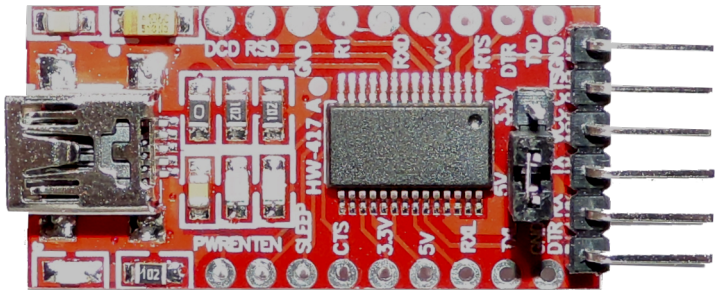

Figure 5: USB to TTL Serial modulSource: own picture

In the picture 5 you see a USB to TTL Serial modul that has a Jumper, where you can select the voltage (3 or 5 v). Please put it on the correct slot, that on VCC are 5 V output voltage.

2 Installation of the required Software

This section describes the required steps, how the ESP-IDF can be installed under Linux.

2.1 Prerequisites

2.1.1 User needs to be added into dialout or uucp group

The user that works later with the ESP32S-CAM board should communicate over USB with it. For this it is required that the user get read and write access to the serial port. On some Linux distributions, the group is named `dialout`, on some other linux distributions, it is `uucp`.

- Ubuntu, Debian, CentOS 7 & 8:

```
$ sudo usermod -a -G dialout $USER
```

- Arch:

```
$ sudo usermod -a -G uucp $USER
```

[1]

2.1.2 Install of required libraries

First of all you have to install the required dependencies on your linux distribution.

- Ubuntu and Debian:

```
$ sudo apt-get install git wget flex bison gperf python3
   ↪ python3-pip python3-setuptools cmake ninja-build
   ↪ ccache libffi-dev libssl-dev dfu-util libusb-1.0-0
```

- CentOS 7 & 8:

```
$ sudo yum -y update && sudo yum install git wget flex
   ↪ bison gperf python3 python3-pip python3-setuptools
   ↪ cmake ninja-build ccache dfu-util libusbx
```

- Arch:

```
$ sudo pacman -S --needed gcc git make flex bison gperf
   ↪ python-pip cmake ninja ccache dfu-util libusb
```

2.2 Get ESP-IDF

To download the ESP-IDF, run the following command in a terminal:

```
$ mkdir -p ~/esp
$ cd ~/esp
$ git clone --recursive https://github.com/espressif/esp-idf.
   ↪ git
```

[2]

2.2.1 Install the ESP-IDF tools

Now you can run the following command in a terminal to install the tools of ESP:

```
$ cd ~/esp/esp-idf
$ ./install.sh esp32
```

[4]

2.3 Set up of the environment variables

The setup does not automatically set up the PATH environment variable. This section describes how to set the environment variables temporary or as an alias.

2.3.1 Set up environment variables temporary

Below you can enter in a terminal the following line to setup the environment variables of the esp idf temporary:

```
$ . $HOME/esp/esp-idf/export.sh
```

Don't forget the space between the leading dot and the path! [3]

2.3.2 Set up environment variables as an alias

If you want to create an alias for your shell profile (`.profile`, `.bashrc`, `.zprofile`, etc), you can add the following line:

```
alias get_idf='. $HOME/esp/esp-idf/export.sh'
```

Now you have to restart the terminal session or you can execute `source [path to profile]` to refresh the configuration.[3]

3 Hardwaresetup

This section describes the setup of the hardware. It should show you, how the boards are connected together.

In picture 6 you can see the setup of the ESP32 CAM Board connected to the USB-to-TTL module.

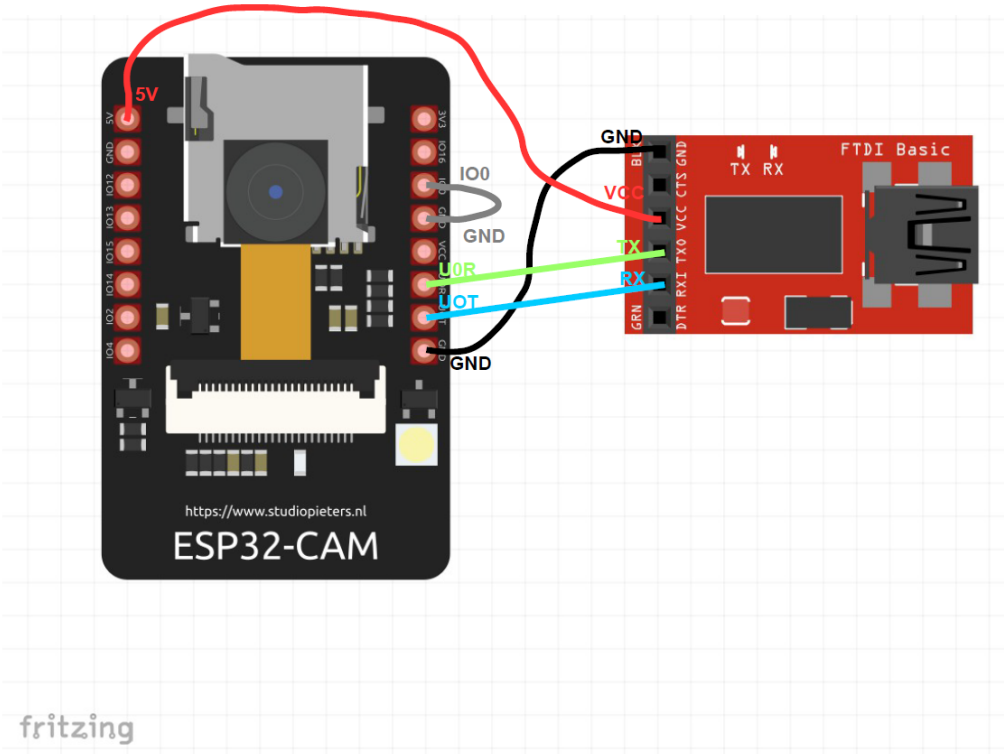

Figure 6: ESP32 CAM Board connected with USB-to-TTL module
Origin Source: https://github.com/AchimPieters/Fritzing-Custom-Parts/blob/master/Fritzing%20Parts/ESP32-CAM_FRONT_2.png
from Achim Pieters
License: MIT

Port USB-to-TTL-Module	PORT ESP32-CAM	PORT ESP32-CAM
GND	GND	-
VCC	5V	-
TX	U0R	-
RX	U0T	-
-	GND	IO0

Table 2: Connections between the ports

For the flashing mode it is needed that IO0 is connected with ground!

4 First steps

4.1 Get Started

1. Connect the USB-to-TTL module together with the ESP32S-CAM board to one free USB-Port on your pc. The setup of the board-connections, can you see in picture 6.

2. Check which port has this USB-to-TTL module get from your OS. You can check this with `ls -lisa /dev/ttyUSB*` (here in our example it is `/dev/ttyUSB0`).

3. Run in a terminal `get_idf`, what we have introduce in section Set up environment variables as an alias.

4. Create in a terminal the folder-structure, where you want to add the hello_world-example. Here we put the hello_world-example under `/esp`:

```
$ mkdir ~/esp
$ cd ~/esp
$ cp -r $IDF_PATH/examples/get-started/hello_world .
```

5. Now to compile the hello_world-sources, run `idf.py build`

```
$ idf.py build
Executing action: all (aliases: build)
...
-- Configuring done
-- Generating done
...
build/bootloader
[874/884] Performing build step for 'bootloader'
[1/3] Linking C executable bootloader.elf
[2/3] Generating binary image from built executable
esptool.py v3.3-dev
Creating esp32 image...
Merged 1 ELF section
Successfully created esp32 image.
...
or run 'idf.py␣-p␣(PORT)␣flash'
```

```
$
```

6. Run as last step the flash-command:

```
$ idf.py -p /dev/ttyUSB0 flash
Executing action: flash
...
hello_world.bin binary size 0x29840 bytes. Smallest app
    ↪ partition is 0x100000 bytes. 0xd67c0 bytes (84%)
    ↪ free.
[2/5] Performing build step for 'bootloader'
...
esptool.py v3.3-dev
Serial port /dev/ttyUSB0
Connecting....
Chip is ESP32-D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in
    ↪  efuse, Coding Scheme None
Crystal is 40MHz
MAC: 78:21:84:7d:01:78
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00007fff...
Flash will be erased from 0x00010000 to 0x00039fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 25424 bytes to 15904...
Writing at 0x00001000... (100 %)
Wrote 25424 bytes (15904 compressed) at 0x00001000 in 0.8
    ↪ seconds (effective 240.1 kbit/s)...
Hash of data verified.
Compressed 170048 bytes to 89645...
Writing at 0x00010000... (16 %)
Writing at 0x0001b0ec... (33 %)
Writing at 0x0002087d... (50 %)
```

```
Writing at 0x00026035... (66 %)
Writing at 0x0002e652... (83 %)
Writing at 0x00036a4a... (100 %)
Wrote 170048 bytes (89645 compressed) at 0x00010000 in 2.5
    ↪  seconds (effective 552.7 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1
    ↪ seconds (effective 307.1 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
Done
$
```

7. If you got the following,

```
$ idf.py -p /dev/ttyUSB0 flash
Executing action: flash
...
esptool.py v3.3-dev
Serial port /dev/ttyUSB0
Connecting.....................
...
$
```

you have to press the reset-button (see here 7 or here 3), that the board goes into the flash-mode. Then it starts to flash the ESP32S-CAM Board [2].

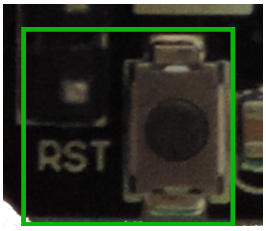

Figure 7: ESP32-S CAM Reset ButtonSource: own picture

4.2 Test of Hello World Example

1. Disconnect the USB-Cable from the USB-to-TTL-Board and the pc, that the boards are currentless.
2. Remove the cable between GND and IO0.
3. Connect again the USB-Cable to the USB-to-TTL-Board.
4. Now open again a terminal and run `get_idf`.
5. Our Example is printing Hello World on terminal over a serial connection. Below you will see the command to open the serial port and also the successful output:

```
$ idf.py -p /dev/ttyUSB0 monitor
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by
    ↪  Ctrl+H ---
Restarting in 3 seconds...
Restarting in 2 seconds...
Restarting in 1 seconds...
Restarting in 0 seconds...
Restarting now.
ets Jun 8 2016 00:22:57

rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0
    ↪ x00,wp_drv:0x00
mode:DIO, clock div:2
```

```
load:0x3fff0030,len:6752
load:0x40078000,len:14796
load:0x40080400,len:3792
0x40080400: _init at ??:?

entry 0x40080694
I (27) boot: ESP-IDF v5.0-dev-1452-g93106c9348 2nd stage
   ↪ bootloader
I (27) boot: compile time 16:08:09
I (27) boot: chip revision: 1
I (31) boot_comm: chip revision: 1, min. bootloader chip
   ↪ revision: 0
I (39) boot.esp32: SPI Speed : 40MHz
I (43) boot.esp32: SPI Mode : DIO
I (48) boot.esp32: SPI Flash Size : 2MB
I (52) boot: Enabling RNG early entropy source...
I (58) boot: Partition Table:
I (61) boot: ## Label Usage Type ST Offset Length
I (69) boot: 0 nvs WiFi data 01 02 00009000 00006000
I (76) boot: 1 phy_init RF data 01 01 0000f000 00001000
I (83) boot: 2 factory factory app 00 00 00010000 00100000
I (91) boot: End of partition table
I (95) boot_comm: chip revision: 1, min. application chip
   ↪ revision: 0
I (102) esp_image: segment 0: paddr=00010020 vaddr=3
   ↪ f400020 size=0782ch ( 30764) map
I (122) esp_image: segment 1: paddr=00017854 vaddr=3
   ↪ ffb0000 size=0243ch ( 9276) load
I (126) esp_image: segment 2: paddr=00019c98 vaddr
   ↪ =40080000 size=06380h ( 25472) load
I (140) esp_image: segment 3: paddr=00020020 vaddr=400
   ↪ d0020 size=14978h ( 84344) map
I (171) esp_image: segment 4: paddr=000349a0 vaddr
   ↪ =40086380 size=04e64h ( 20068) load
I (180) esp_image: segment 5: paddr=0003980c vaddr
   ↪ =50000000 size=00010h ( 16) load
I (186) boot: Loaded app from partition at offset 0x10000
I (186) boot: Disabling RNG early entropy source...
```

```
I (202) cpu_start: Pro cpu up.
I (202) cpu_start: Starting app cpu, entry point is 0
    ↪ x40081004
0x40081004: call_start_cpu1 at cpu_start.c:152

I (189) cpu_start: App cpu up.
I (217) cpu_start: Pro cpu start user code
I (217) cpu_start: cpu freq: 160000000 Hz
I (217) cpu_start: Application information:
I (221) cpu_start: Project name: hello_world
I (227) cpu_start: App version: 1
I (231) cpu_start: Compile time: Feb 11 2022 17:57:03
I (237) cpu_start: ELF file SHA256: 273a35a73851882c...
I (243) cpu_start: ESP-IDF: v5.0-dev-1452-g93106c9348
I (250) heap_init: Initializing. RAM available for dynamic
    ↪  allocation:
I (257) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (263) heap_init: At 3FFB2D30 len 0002D2D0 (180 KiB):
    ↪ DRAM
I (269) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/
    ↪ IRAM
I (276) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/
    ↪ IRAM
I (282) heap_init: At 4008B1E4 len 00014E1C (83 KiB): IRAM
I (289) spi_flash: detected chip: generic
I (293) spi_flash: flash io: dio
W (297) spi_flash: Detected size(4096k) larger than the
    ↪ size in the binary image header(2048k). Using the
    ↪ size in the binary image header.
I (311) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Hello world!
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE,
    ↪ silicon revision 1, 2MB external flash
Minimum free heap size: 296120 bytes
...
```

5 Workarounds

5.1 Still connection error after press reset button

If you run the flash command and you got a Timed Out

```
$ idf.py -p /dev/ttyUSB0 flash
Executing action: flash
...
esptool.py v3.3-dev
Serial port /dev/ttyUSB0
Connecting.....................
...

A fatal error occurred: Failed to connect to ESP32: Timed out
    ↪ waiting for packet header
$
```

you should check if you have problems with the 5V output on your FTDI-board. If yes, you should modify your setup by an external power supply on the 5V pin on the board (see 8).

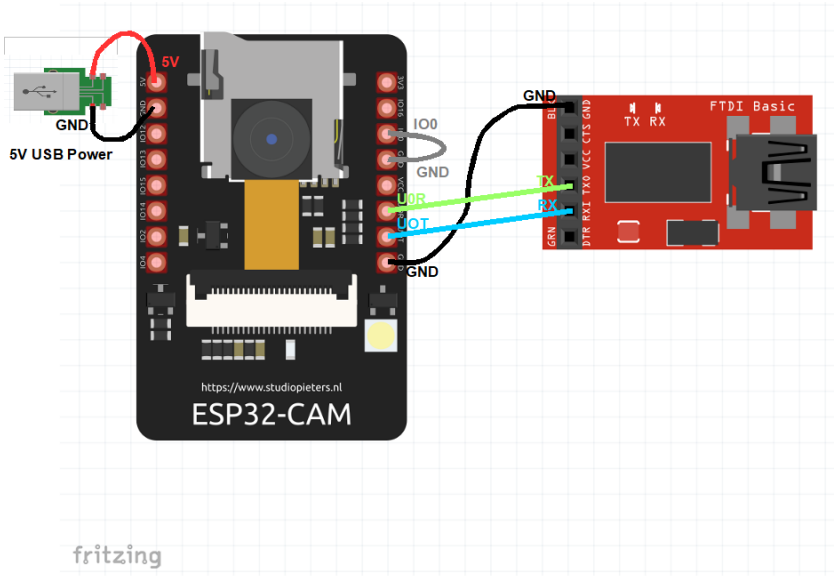

Figure 8: ESP32S CAM setup with external 5V power supply
Source: own picture

6 Additional Links

- Additional Tips for ES32 CAM Troubleshooting
- Wasserzähler mit ESP-CAM auslesen / AI-on-the-edge from haus-automatisierung.com (Matthias Kleine)
- Getting Strated with Bare Metal ESP32 Programming

References

[1] Espressif. *Establish Serial Connection with ESP32 - Adding user to dialout on Linux*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/establish-serial-connection.html. Online; accessed 11 Februrary 2022.

[2] Espressif. *Get ESP-IDF - Linux and macOS*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html. Online; accessed 06 Februrary 2022.

[3] Espressif. *Set up the environment variables - Linux and macOS*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html. Online; accessed 06 Februrary 2022.

[4] Espressif. *Set up the tools - Linux and macOS*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html. Online; accessed 06 Februrary 2022.

[5] Unknown. *FEATURES & SPECIFICATIONS*. http://esp32.net. Online; accessed 06 Februrary 2022.

[6] Unknown. *Product Specifications*. https://loboris.eu/ESP32/ESP32-CAM%20Product%20Specification.pdf. Online; accessed 06 Februrary 2022.

[7] SmartHome yourself. *ESP32Cam als extrem günstige IP KAMERA nicht nur für HOME ASSISTANT! [Video]*. https://www.youtube.com/watch?v=qVDTlPSCJ8E&t=720s. Online; accessed 06 Februrary 2022.