Open in app ↗ Sign up Sign in

Medium

Reverse engineering of esp32 flash dumps with ghidra or IDA Pro



Olof Astrand · Follow

5 min read · Apr 20, 2021

Listen Share

25



The most popular story I have written so far on medium, was about about analyzing an esp32 fash dump that I created myself. https://olof-astrand.medium.com/analyzing-an-esp32-flash-dump-with-ghidra-e70e7f89a57f

One of the drawbacks of this procedure is that you need to build the esp32 flash loader as a plugin for ghidra. This is something you want to avoid. So in this story, I will go through another method, where you re-create an Elf file from the flash file instead. This Elf file can then be analyzed with Ghidra or IDA Pro which are reverse engineering tools, that both have the possibility to partially re-create the source code.

This method has been done before, and I found this on the internet:

https://youtu.be/w4_3vwN_2dI . It is a talk that describes te technique. Here is the python source. https://github.com/tenable/esp32_image_parser

You still need to install a plugin for your tool, such as https://github.com/Ebiroll/ghidra-xtensa for ghidra (unless you use ghidra version 11,0 or newer) , or https://github.com/themadinventor/ida-xtensa for IDA, in order to get them to understand the Xtensa esp32 instructions. In this story, we will be using Ghidra V9.2.2 public with the ghidra-xtensa plugin.

Ghidra version 11.0 includes xtensa support, so the information here should work out of the box.

As a flash dump, I will look at a preloaded flash file that came from China with a esp32 “Wemos” device that I bought in 2017.

The first step is to dump the flash content.

```
esptool.py -p /dev/ttyUSB0 -b 460800 read_flash 0 0x400000
flash.bin
```

After installing the esp32_image_parser you can show the partitions.

```
./esp32_image_parser.py show_partitions wemos.bin
```

After identifying what partition to extract, use the image_parser_tool to create an elf file from the flash dump,

```
python3 esp32_image_parser.py create_elf wemos.bin -partition ota_1 -output
ota_1.elf
```

Verify the generated elf file

```
readelf -h ota_1.elf
```

```
ELF Header:
Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class: ELF32
Data: 2's complement, little endian
Version: 1 (current)
OS/ABI: UNIX – System V
ABI Version: 0
Type: EXEC (Executable file)
Machine: Tensilica Xtensa Processor
Version: 0x1
Entry point address: 0x400811bc
...
```

After starting up qemu I will set gdb to this entrypoint as breakpoint. It will also be used by ghidra for the point where it starts to look for all functions.

Identifying the functions

I will use 2 different techniques to identify the functions. 1) by running the dumped flash image in qemu and 2) by comparing the startup code with my own compiled elf file.

Startup qemu,

```
xtensa-softmmu/qemu-system-xtensa -M esp32 -s -d guest_errors,page
-nographic -drive file=wemos.bin,if=mtd,format=raw -global
driver=timer.esp32.timg,property=wdt_disable,value=true -no-reboot
-s -S
```

At startup the rom will load the bootlader and output the following,

```
Adding SPI flash device
ets Jul 29 2019 12:21:46

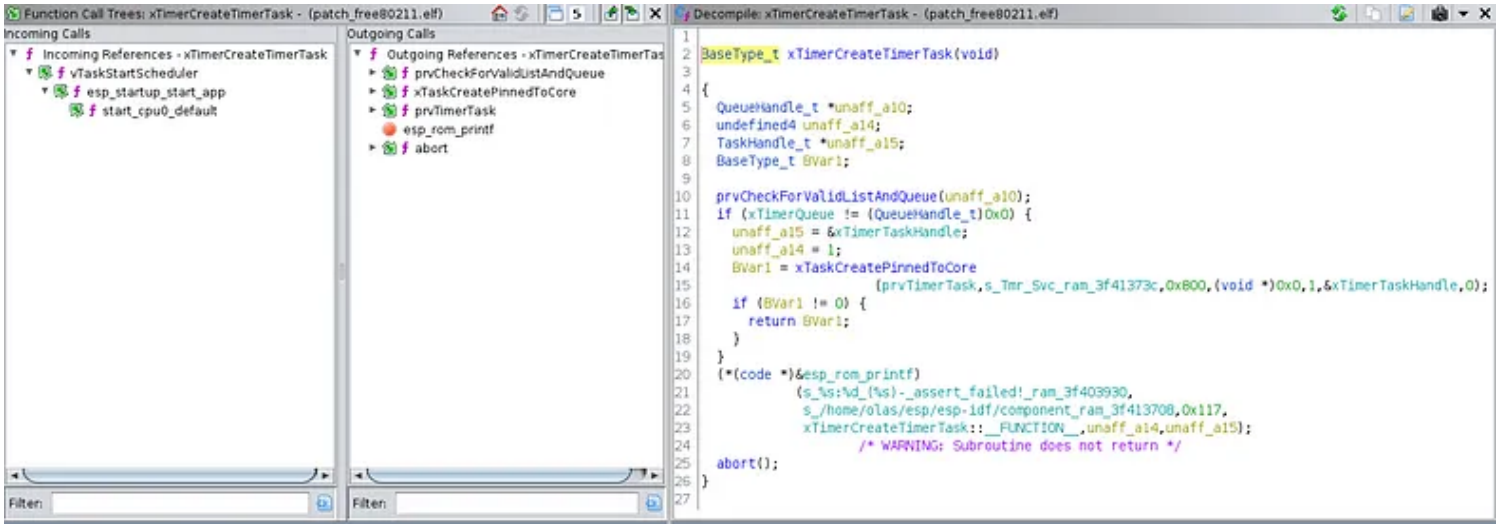
rst:0x1 (POWERON_RESET),boot:0x12 (SPI_FAST_FLASH_BOOT)
M25P80: Read id (command 0x90/0xAB) is not supported by device
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv
:0x00
mode:DIO, clock div:1
load:0x3fff0008,len:8
load:0x3fff0010,len:160
```

```
load:0x40078000,len:10632
load:0x40080000,len:252
entry 0x40080034
M25P80: Unknown cmd 31
```

The M25P80 output is the emulated flash complaining about unimplemented functions. So the espressif qemu version unfortunately did not work in this case, so instead I used this https://github.com/Ebiroll/qemu_esp32

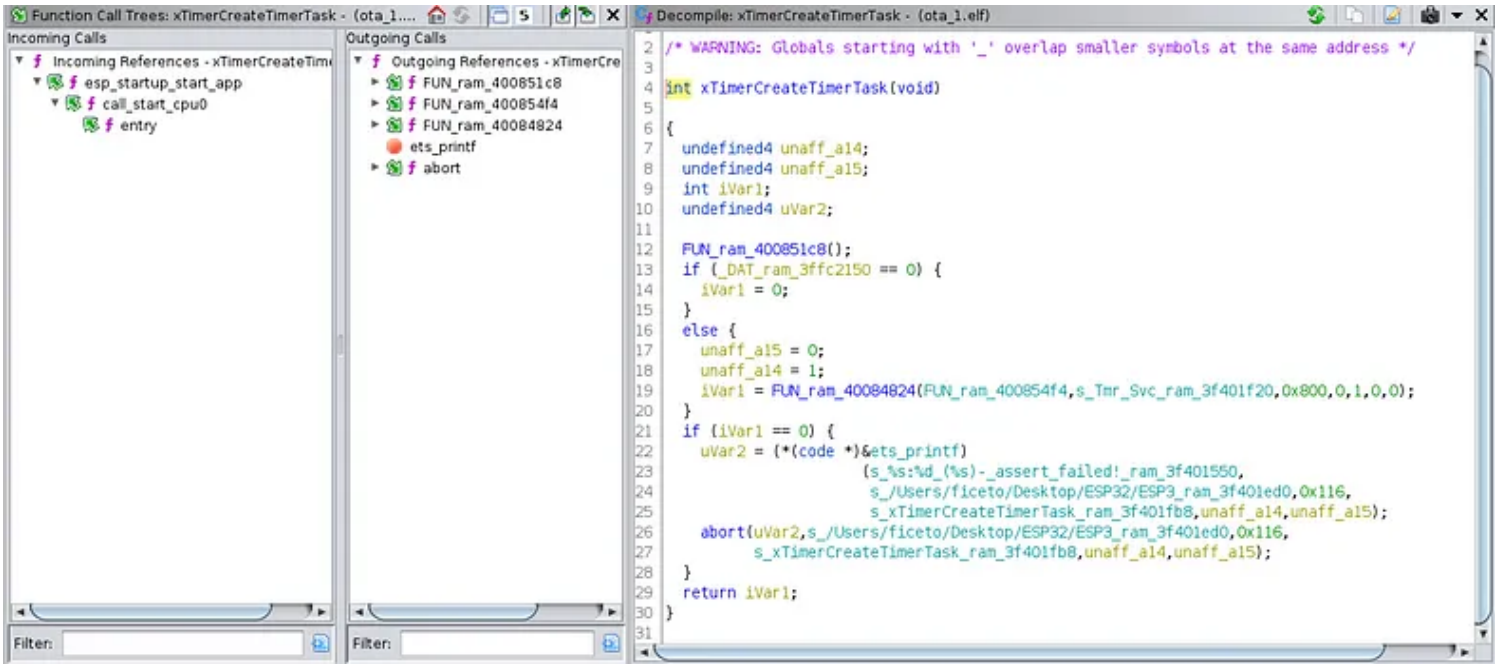
If you compare the structure between an known elf file and the one from the flash dump, it is not so hard to rename the FUN_ram_4008XXX functions. You should also make sure to use the assert printouts, that normally tells you the function name.

At startup, the entry() function is called from the bootloader, from there we will try to identify the xTaskCreate and xTaskCreatePinnedToCore functions.



Startup in known elf

We compare this with our function from the flash dump



xTimerCreateTimerTask

An other important goal is to find the main task.

```
undefined4 main_task(undefined4 param_1)

{
 uint uVar1;
 undefined4 uVar2;

 uVar1 = _DAT_ram_3ff4808c;
 memw();
 _DAT_ram_3ff5f048 = _DAT_ram_3ff5f048 & 0xfffffff1;
 memw();
 memw();
 _DAT_ram_3ff4808c = _DAT_ram_3ff4808c & 0xfffffbff;
 memw();
 uVar2 = FUN_ram_40086098(uVar1);
 app_main(uVar2);
 vTaskDelete(0);
 return param_1;
}
```

The main task calls app_main() and there, usually the magic happens.

However in this case, app_main just creates a task that I called loop_task()

```
undefined4 app_main(undefined4 param_1)

{
 undefined4 unaff_a10;

 setup(unaff_a10);
 xTaskCreatePinnedToCore(loop_task,s_?loopTask_ram_3f40324b +
1,0x1000,0,1,0,1);
 return param_1;
}
```

Here I have roughly identified what the loop task does,

loop_task

We now set a breakpoint here, to get some help in understanding the handle_tick() function.

Start the esp32 qemu and the debugger, then set a breakpoint in the loop function and the add_text function.

```
xtensa-softmmu/qemu-system-xtensa -M esp32 -s -d guest_errors,page
-nographic -drive file=wemos.bin,if=mtd,format=raw -global
driver=timer.esp32.timg,property=wdt_disable,value=true -no-reboot

xtensa-esp32-elf-gdb.qemu -ex 'target remote:1234' ota_1.elf

(gdb) b *0x400d7a90
(gdb) layout asm
(gdb) si
```

```
(gdb) p (char *) $a3
$9 = 0x3f400bcb "Dec 23 2017 10:27:52"
```

The inner loop

When we run continue and the the program in qemu, it outputs,

```
Hello, My firstESP32 !
Dec 23 2017 10:27:52
```

Conclusion

We were enable to reconstruct the program from the flashfile, with help from ghidra and the esp32_image_parser programs. In this case, it seems like the preprogrammed flash contained an Arduino compiled application, that outputs, Hello, My firstESP32 !. We did not use qemu so much but it can be helpful when trying to understand the details of the program. Also make sure to use cross references [XREF] of known strings to find names of functions. Rename the functions and try to assign the correct arguments, as it greatly helps you and

ghidra to analyze the program.

Esp32 Ghidra Reverse Engineering

Follow

Written by Olof Astrand

22 Followers

Software Engineer.

More from Olof Astrand





Olof Astrand

Running Ghidra Debugger IN-VM

Starting up ghidras debugger tool can be a bit confusing the first time you use it. Here I will go through how to do it first with a local...

4 min read · Jan 21, 2024







Olof Astrand

Analyzing an esp32 flash dump with ghidra

As a third step I will use the flash loader to import the same example, as the previous example...

6 min read · Aug 3, 2020


1

1




Olof Astrand

Hacking wireless sockets like a NOOB

Learn how to use the Universal Radio Hacker, software, in order to analyze the signals used to control a wireless socket device.

10 min read · Apr 11, 2021

8





Olof Astrand

Ghidra debug tool with Qemu

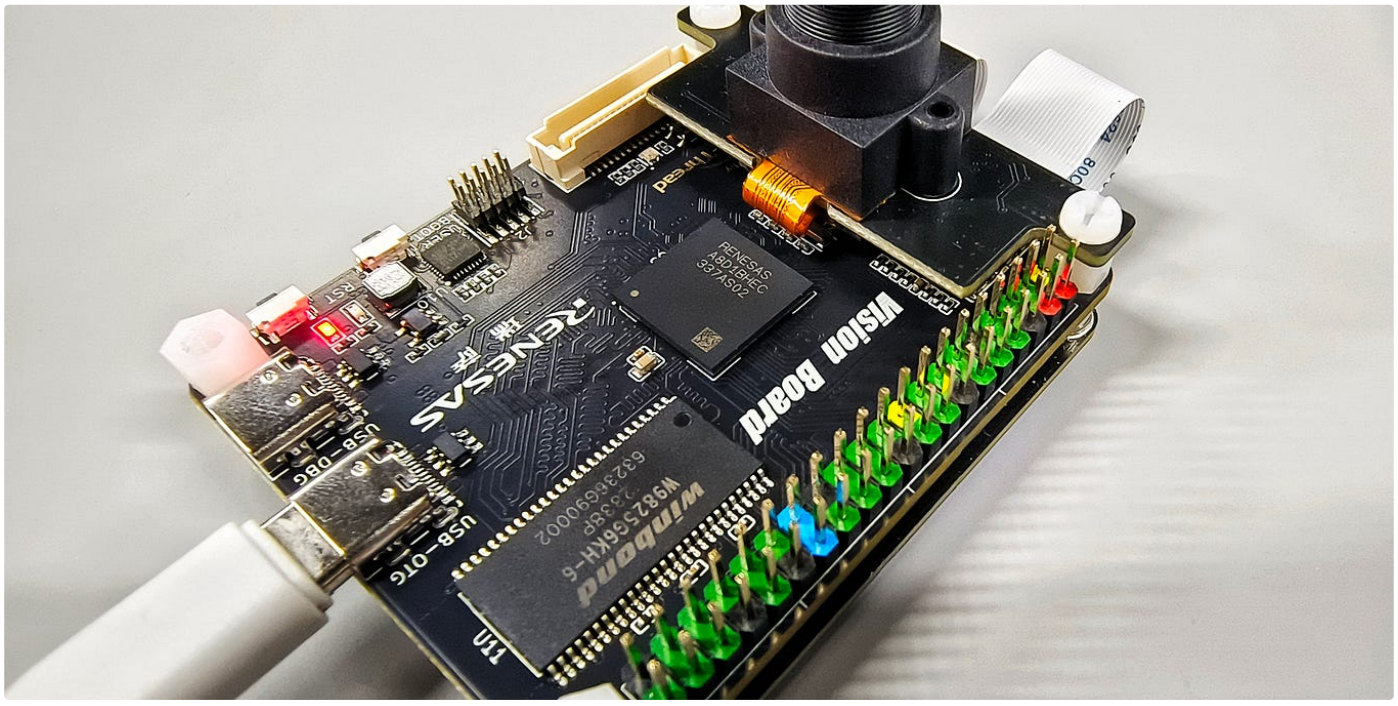
Here we will look at debugging binaries on a remote target with qemu. First I try the ghidra debug tool, without success. So we try...

9 min read · Jan 22, 2024

90

See all from Olof Astrand

Recommended from Medium



RT-Thread IoT OS

RT-Thread IoT OS

Pre-Order: ARM Cortex-M85 Vision Board

We’re thrilled to announce that RT-Thread in collaboration with Renesas Electronics, is unveiling its latest hardware innovation—Vision...

3 min read · Mar 12, 2024

3



Olof Astrand

Running Ghidra Debugger IN-VM

Starting up ghidras debugger tool can be a bit confusing the first time you use it. Here I will go through how to do it first with a local...

4 min read · Jan 21, 2024

Lists



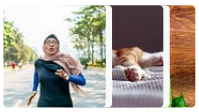
Staff Picks

630 stories · 920 saves



Stories to Help You Level-Up at Work

19 stories · 580 saves



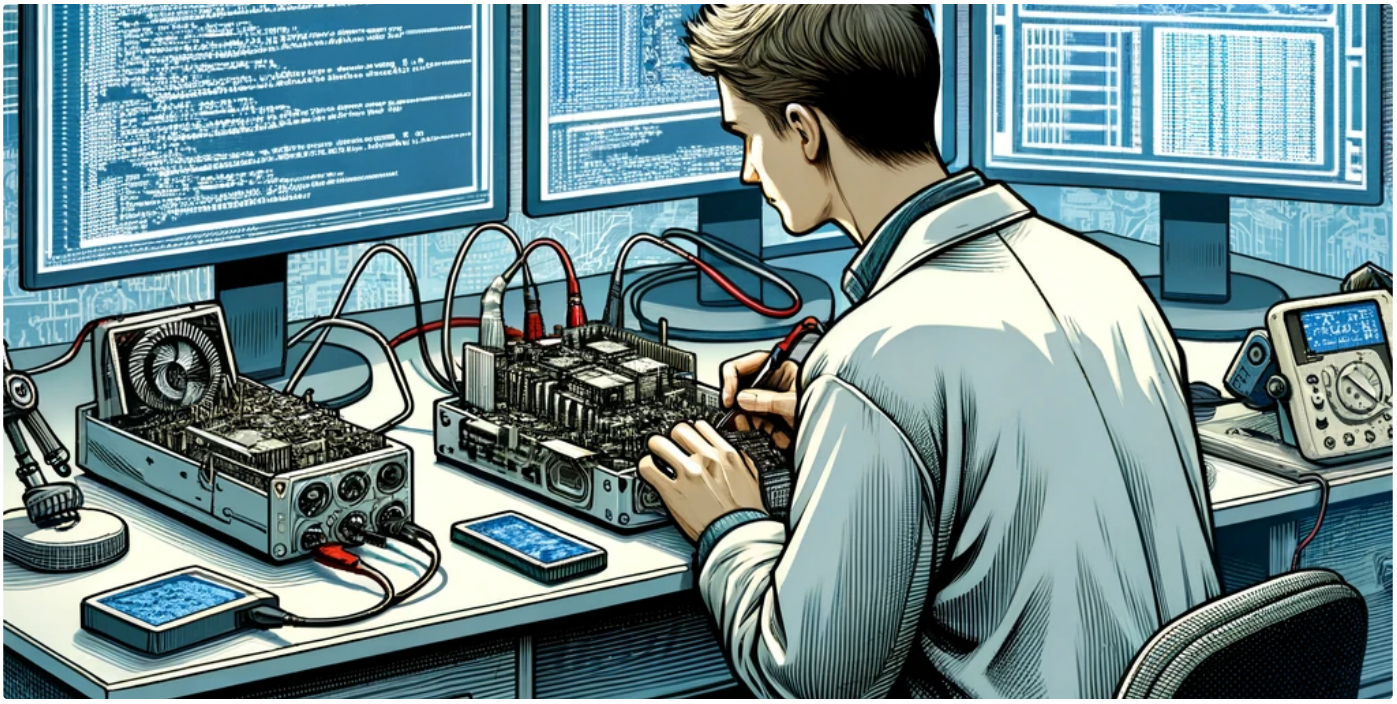
Self-Improvement 101

20 stories · 1673 saves



Productivity 101

20 stories · 1540 saves



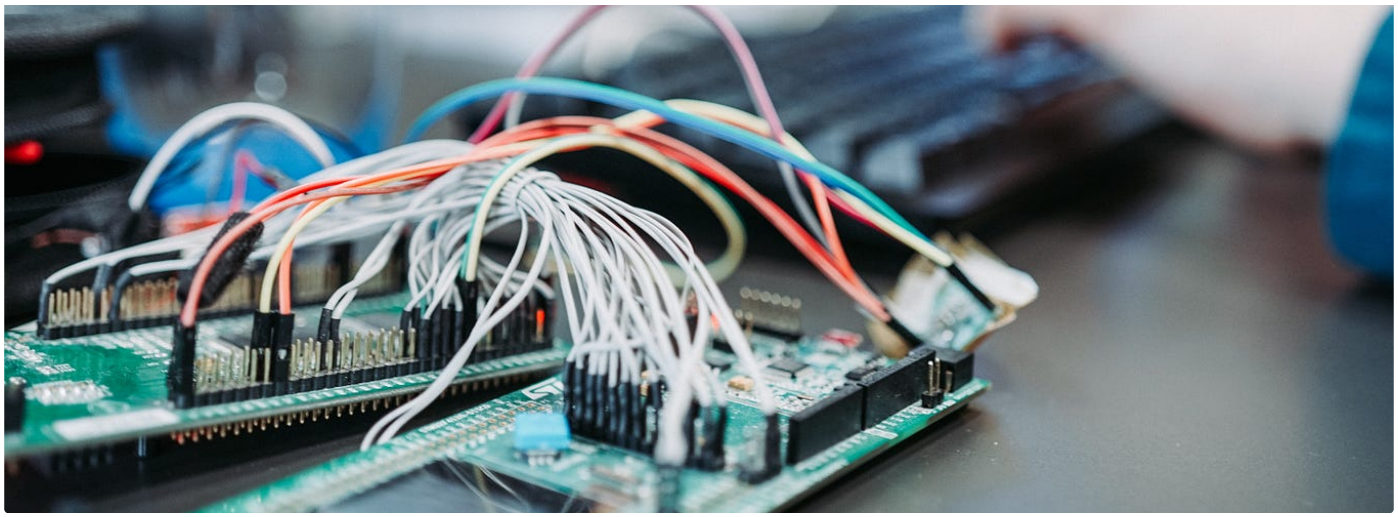
Path Cybersec [Slava Moskvin]

Extracting Firmware: Every Method Explained

8 min read · Jan 2, 2024

84





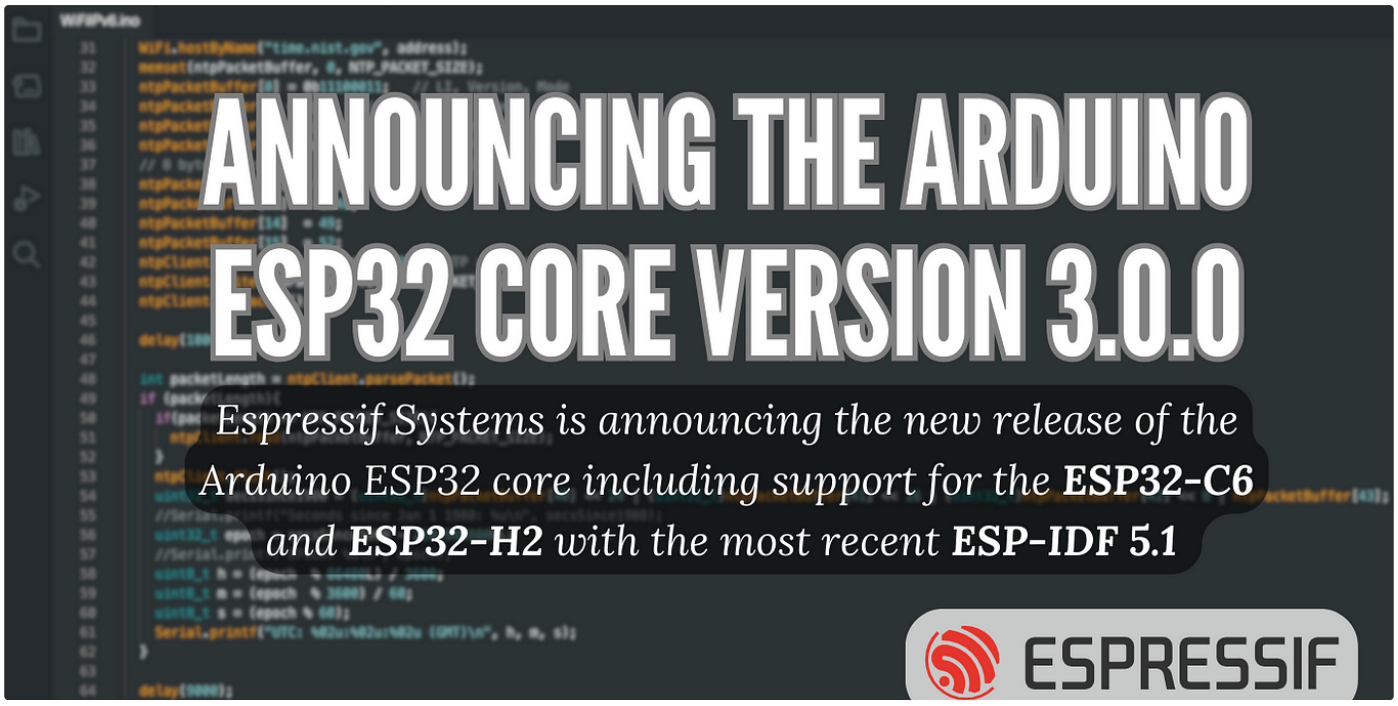
Punnyarthabanerjee

ESP8266: Handling data in JSON

If you need to handle your data in JSON on your next IOT project, this is the tutorial for you.

5 min read · Dec 15, 2023

53



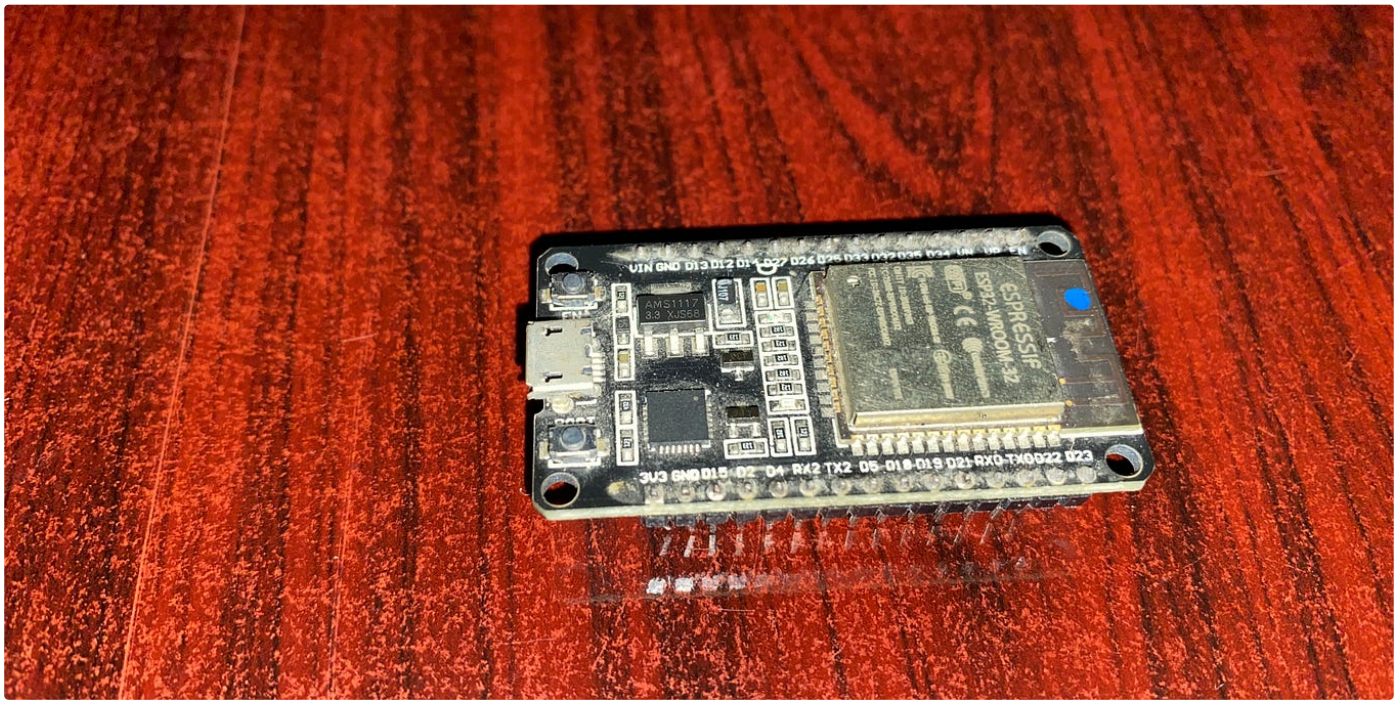
Pedro Minatel in The ESP Journal

Announcing the Arduino ESP32 Core version 3.0.0

Espressif Systems is announcing the new release of the Arduino ESP32 core including

6 min read · Nov 2, 2023

121



Samuel Adesola

How to set up ESP32-WROOM-32

Hey guys, this is going to be a quick guide on how to set up the ESP32-WROOM-32 board. This method is also applicable to some other ESP32...

4 min read · Dec 16, 2023

3

See more recommendations