

Lecteur RFID UHF V3 éducation

RAPPORT DE MODIFICATIONS

Cruz Pastor, Dietrich, Carballo | ISE-ISC | 14.3.2022

Table des matières

Introduction	2
Recherche de composants et du module WIFI/Bluetooth	2
Modification du schéma électrique	2
Réalisations empreintes	4
Emplacement et routage PCB.....	6
Réalisation et test de fonctionnement.....	8
Conclusion.....	8
Annexe.....	9
GUIDE ESP32-C3-WROOM-02	9

Introduction

Dans ce travail, il nous a été demandé de modifier la carte réalisée par Gaëtan Passeri pour son travail de bachelor, qui est un lecteur RFID UHF.

Les modifications à réaliser sont de rajouter une interface WIFI/Bluetooth, supprimer les points de test superflu et de déplacer l'écran vers le centre de la carte. Dans un second temps nous allons identifier les composants qui ne sont plus disponibles et les replacer par des équivalents. Le schéma et le PCB devront être adaptés avec ces nouveaux composants et dans un souci d'écologie et de réparabilité nous devons choisir des composants conventionnels et simples à remplacer.

Finalement nous allons envoyer notre carte en production, la monter et la tester.

Recherche de composants et du module WIFI/Bluetooth

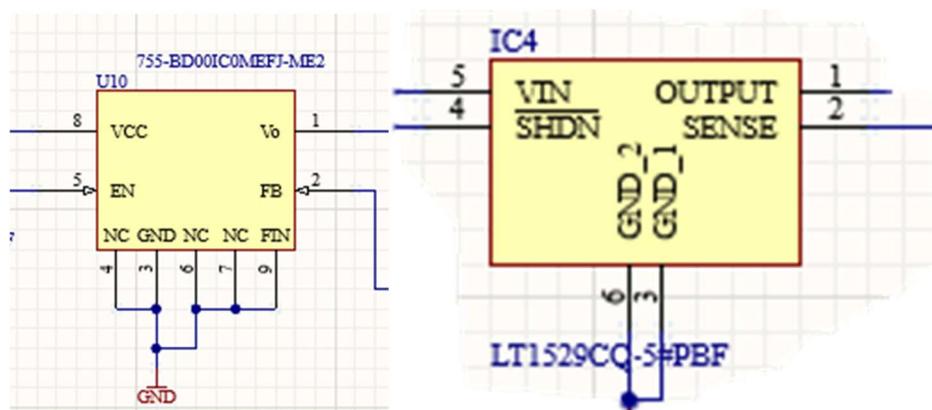
Voici les composants que nous avons dû modifier :

- Des condensateurs 5,6[pF]
- Une diode Schottky
- Des billes de ferrite de 1 [k Ω]
- Des régulateurs de tension de 3,3 et 5 [V]
- Une interface pour la communication UART/USB
- Un condensateur programmable par SPI
- Un quartz de 32,768 [KHz]

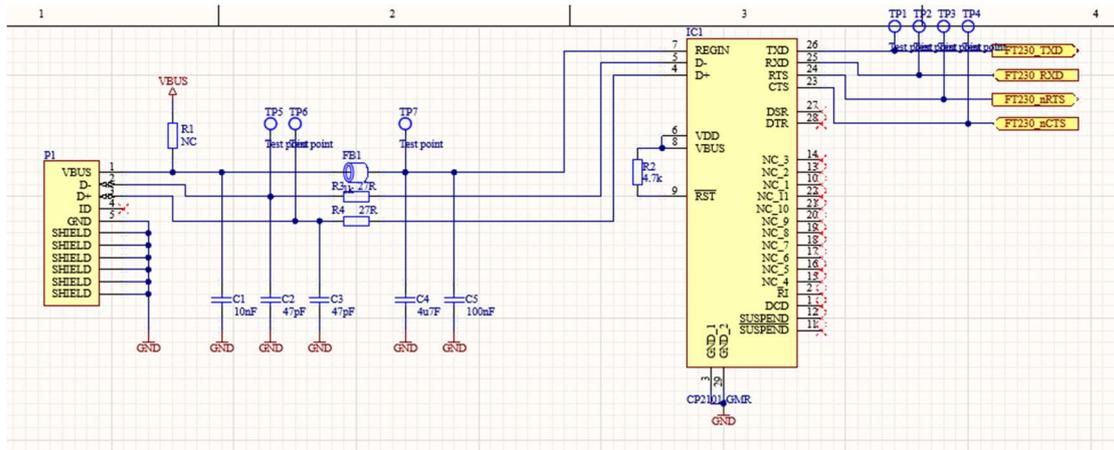
Il faut aussi rechercher un module pour le WIFI/Bluetooth. Pour cela nous avons sélectionné le ESP32-C3-WROOM-02-N4 avec les composants nécessaires pour son bon fonctionnement.

Modification du schéma électrique

Certains composants avaient les mêmes boîtiers que leurs originaux, néanmoins les composants que nous avons dû remplacer sont les régulateurs de tension pour avoir un ampérage plus grand et le pont UART-USB.



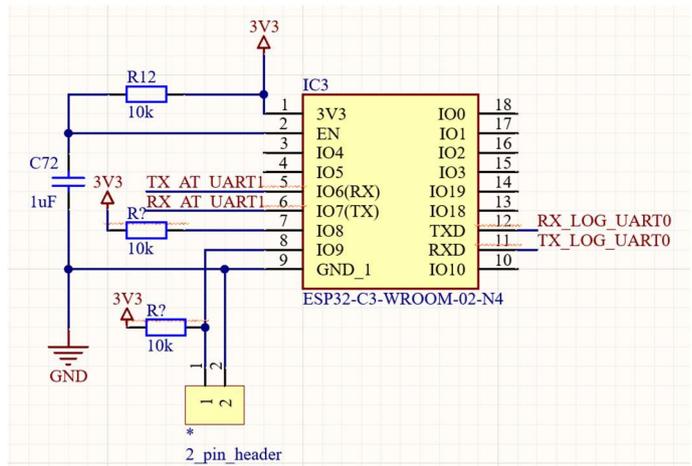
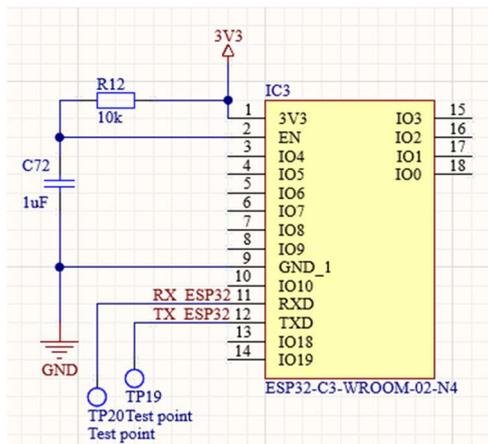
Régulateurs de tension 3.3 (à gauche) et 5 volts.



Pont USB-UART.

Nous avons supprimé un grand nombre de points de test en gardant les principaux. Ensuite nous avons supprimé le régulateur 3.6 [V], car le composant qui l'utilisait accepte le 3.3[V] qui est déjà disponible sur la carte.

Nous avons connecté l'ESP32 au microcontrôleur en passant par un des UART (PIO1_12 TX PIO1_13 TX).



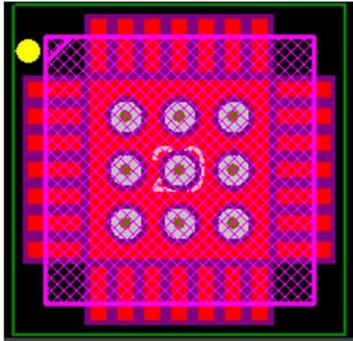
À gauche on peut voir ce qui a été réalisé, mais en testant le module nous avons découvert que l'ESP utilise deux uart (un pour le log et la programmation, le deuxième pour les commandes AT) il faudra refaire la carte en modifiant le schéma comme l'ont fait le groupe de Mathias. Donc pour faire fonctionner la carte déjà réalisée il faut couper les connections des pin 11 et 12 et les re router vers les pins 5 et 6. Il faut aussi connecter la pin 8 au VCC.

Le jumper étant utilisé pour choisir le mode de boot il faut souder un ESP qui est déjà programmé sur notre carte. Vous trouverez la méthodologie de programmation en annexe.

Enfin, pour gagner encore plus de place pour rajouter le module ESP32, nous avons modifié les boutons par d'autres, plus petits.

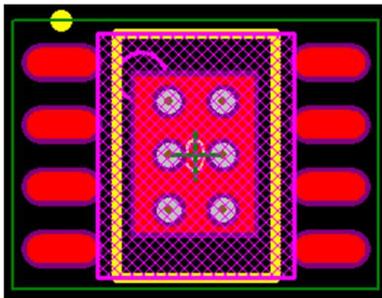
Réalisations empreintes

Pour commencer, nous avons réalisé l'empreinte du pont USB-UART. Afin d'éviter le surplus de patte à souder, nous avons rajouté des vias.



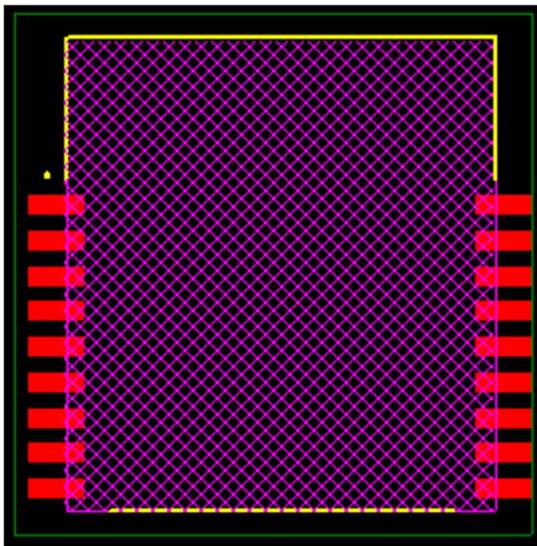
Pont USB-UART

Même chose pour le régulateur de tension 3.3 [V].



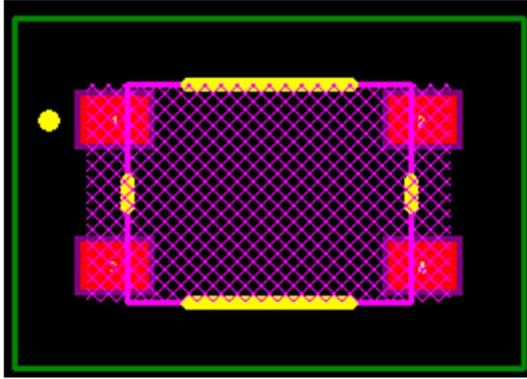
Régulateur de tension 3.3[V]

Pour ce qui est du module ESP 32 nous avons augmenté la taille des pads pour faciliter la réparation.



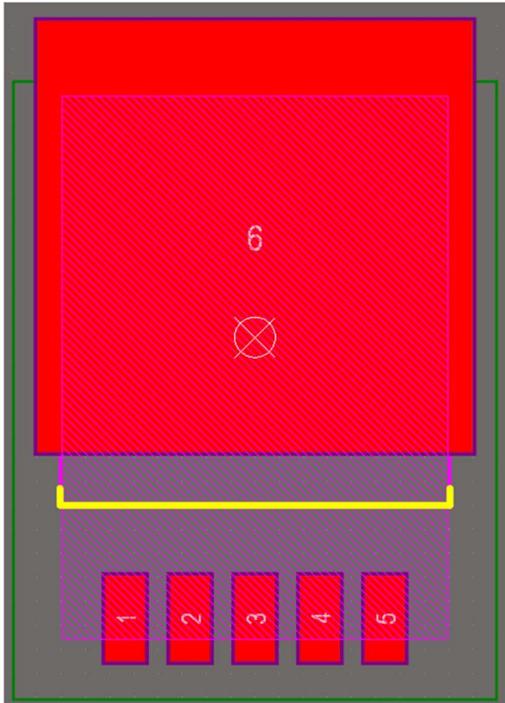
Module ESP32

Voici les boutons-poussoirs que nous avons utilisés



Bouton-poussoir

Pour le régulateur de tension, nous n'avons pas mis de via pour avoir suffisamment de soudure pour dissiper la chaleur. Nous avons néanmoins augmenté la taille du pad pour pouvoir y dessouder plus facilement.

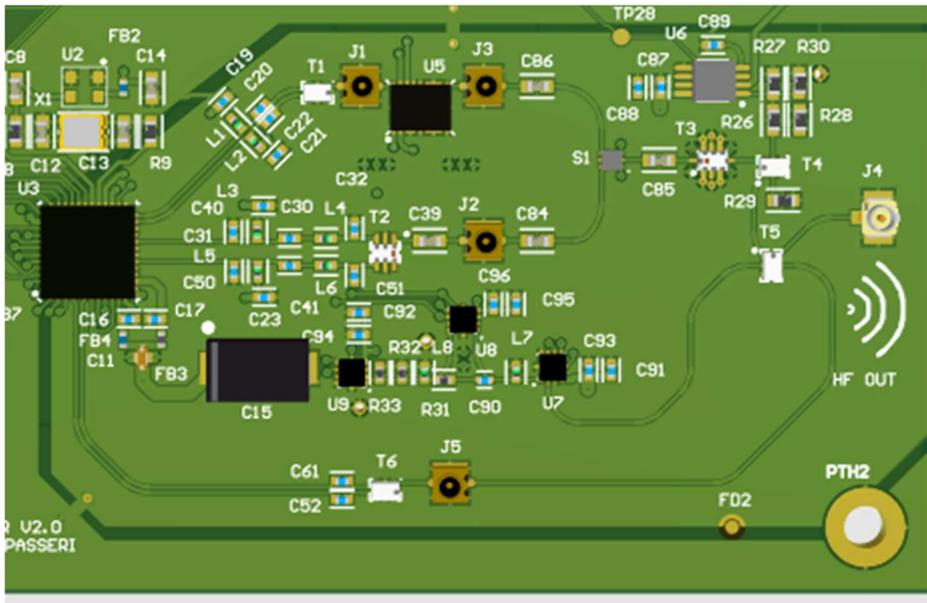


Régulateur de tension 5[V]

Emplacement et routage PCB

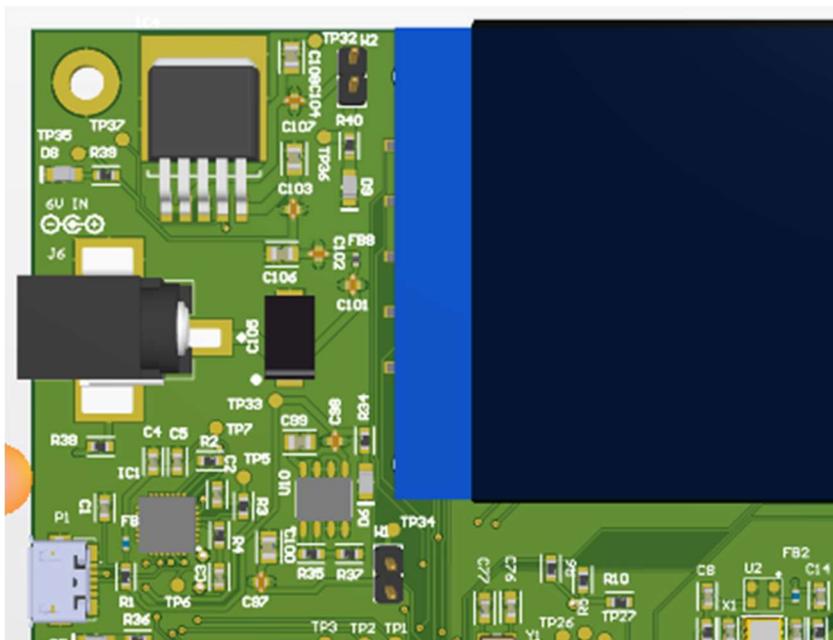
Pour l'emplacement des composants sur la carte, les contraintes suivantes ont dû être respectées :

- La partie RF ne doit pas être modifiée.
- L'écran LCD doit être replacé au centre de la carte.
- Le module ESP32 doit être placé loin de l'alimentation pour éviter des perturbations.



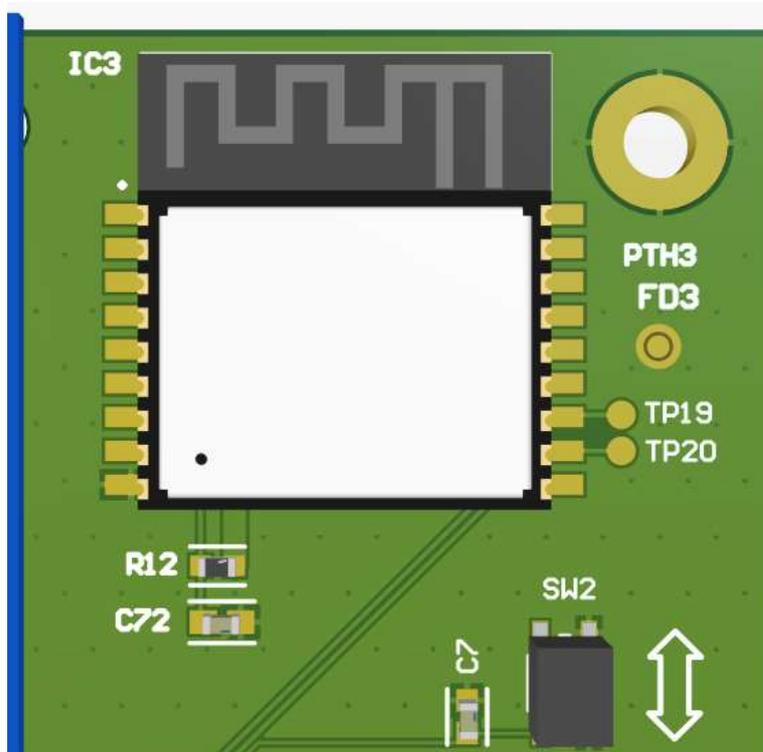
Lecteur RFID

La partie Alimentation a dû être compressée pour accueillir le nouvel emplacement de l'écran.



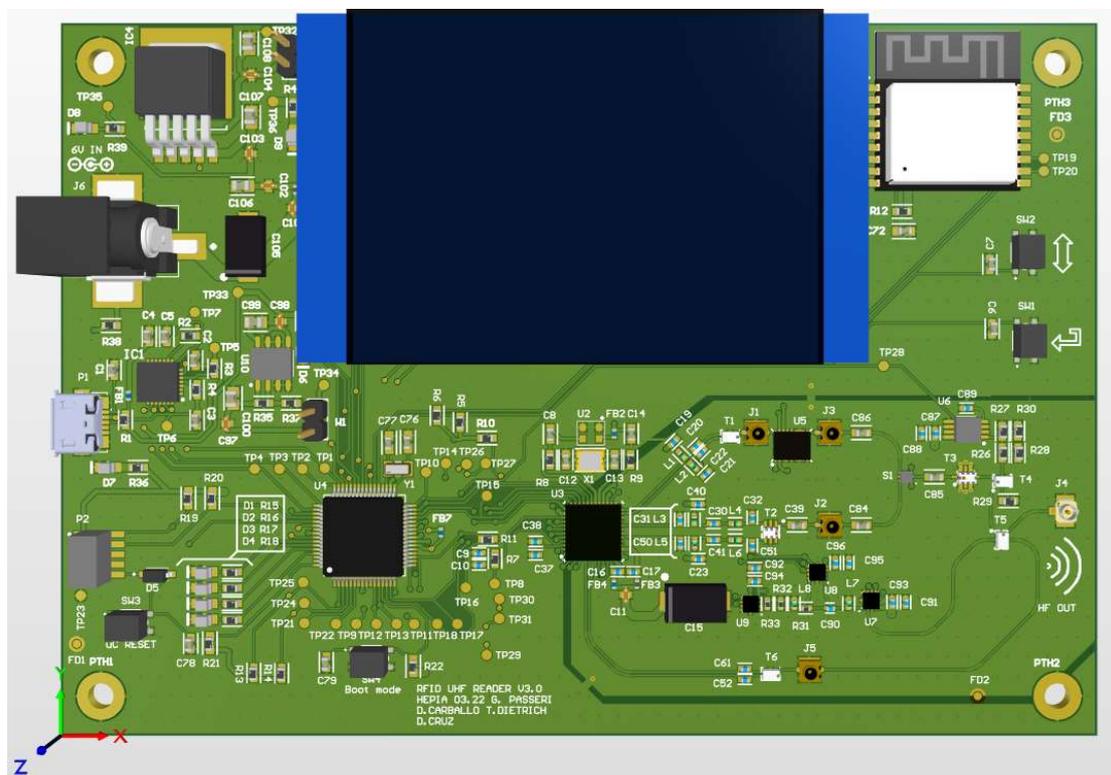
Côté gauche de la carte.

Nous avons placé l'ESP à côté de l'écran, ce qui est le plus éloigné des alimentations.



Emplacement ESP32

Voici donc la carte qui a été envoyée en production.



Vue d'ensemble de la carte

Réalisation et test de fonctionnement

Pour ce qui est de la réalisation, le temps étant trop court, nous avons seulement réalisé la face arrière de la carte.



Conclusion

En conclusion ce qui nous a pris le plus de temps était le routage de la partie alimentation du fait qu'on devait mettre le minimum de composant sur l'autre face. La recherche de composant était aussi difficile, car d'une semaine à l'autre beaucoup de composants n'étaient plus disponibles.

Malheureusement comme écrit auparavant le routage de l'ESP est faux, les erreurs sont réparables, mais il faudrait refaire une version corrigée pour la production finale.

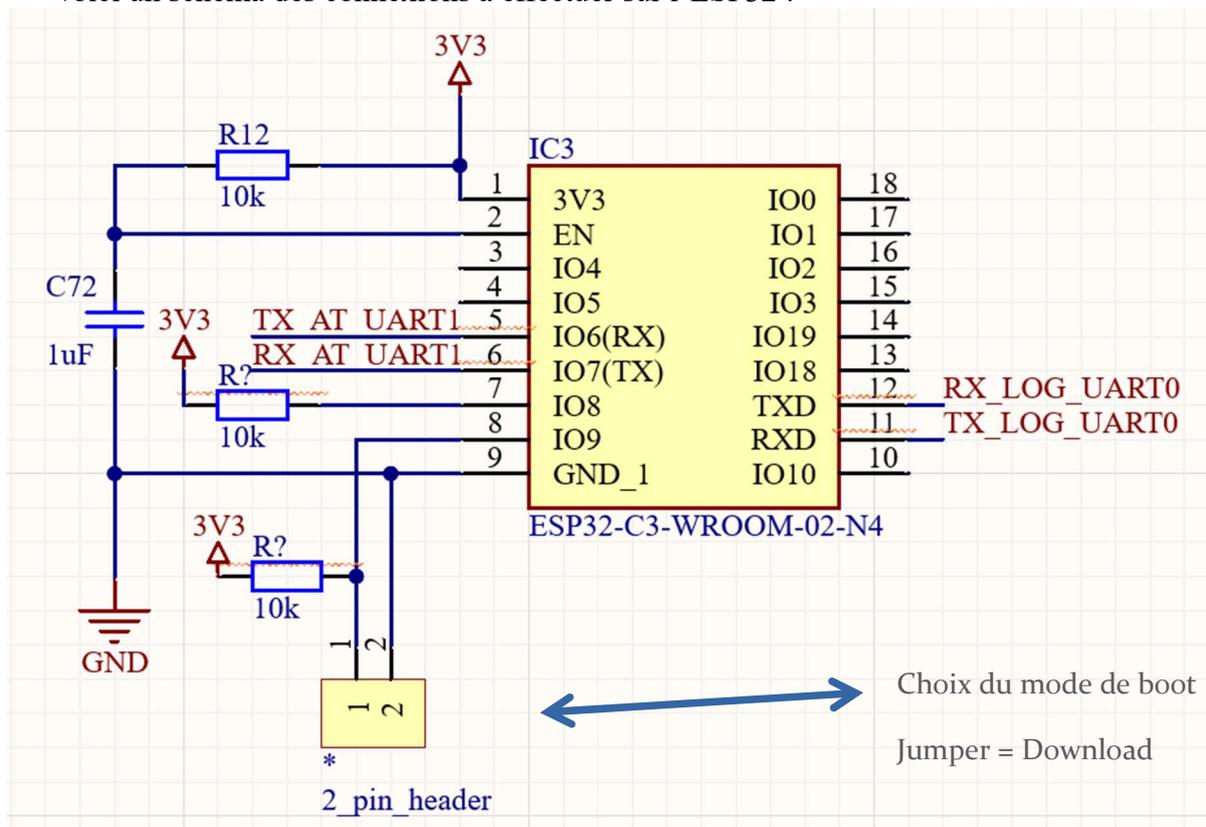
Néanmoins nous avons pu en apprendre plus sur les étapes de conception pour des systèmes embarqués.

Annexe

GUIDE ESP32-C3-WROOM-02

Guide réalisé pour cet ESP32 : 356-ESP32C3WROOM02N4

Voici un schéma des connexions à effectuer sur l'ESP32 :



L'UART 0 sert à flasher l'ESP32 et à récupérer les LOG, alors que l'UART 1 permet de communiquer en utilisant les commandes AT une fois que le programme est flashé. Ces informations viennent de ce tableau disponible dans le guide de l'ESP32.

Table 6: ESP32-C3 Series Hardware Connection Pinout

Function of Connection	ESP Board Pins	Other Device Pins
Download/Log output ¹	UART0 <ul style="list-style-type: none"> • GPIO20 (RX) • GPIO21 (TX) 	PC <ul style="list-style-type: none"> • TX • RX
AT command/response ²	UART1 <ul style="list-style-type: none"> • GPIO6 (RX) • GPIO7 (TX) • GPIO5 (CTS) • GPIO4 (RTS) 	USB to serial converter <ul style="list-style-type: none"> • TX • RX • RTS • CTS

Ensuite l'ESP32 a plusieurs modes de démarrage :

Booting Mode ¹			
Pin	Default	SPI Boot	Download Boot
IO8	N/A	Any value	1
IO9	Pull-up	1	0

Il y a le mode Download qui permet de flasher l'ESP32 avec son firmware, et le mode SPI boot qui permet de démarrer sur le firmware préalablement flashé. À noter que les pins IO8 et IO9 ne doivent pas être mise à 0 en même temps.

Pour commencer, coupez l'alimentation de votre ESP32. Connectez la pin IO8 à 1, et IO9 à 0 afin de la mettre en mode download (branchez le Jumper). Connecter votre adaptateur USB-UART sur l'UART0, la communication UART est de 115200 Bauds. Pour finir allumer l'ESP32 en l'alimentant.

Vous devriez obtenir ce message : (Ou n'importe quel message d'erreur, ex `invalid_header`)

```
ESP-ROM:esp32c3-api1-20210207
Build:Feb 7 2021
rst:0x1 (POWERON),boot:0x4 (DOWNLOAD(USB/UART0/1))
waiting for download
```

Cette étape permet simplement de vérifier que la connexion est fonctionnelle.

Télécharger l'outil esptool à cette adresse :

<https://github.com/espressif/esptool>

Ensuite, ouvrez un terminal dans le dossier, et exécutez la commande suivante :

```
> esptool.py --port /dev/ttyUSB0 erase_flash
```

Voici ce que vous devriez obtenir :

```
1/1 + [f] [m] ...x: tanguy@tanguy-Ubuntu: ~/Bureau/tuto_esp32/esptool
1: tanguy@tanguy-Ubuntu: ~/Bureau/tuto_esp32/esptool
tanguy@tanguy-Ubuntu:~/Bureau/tuto_esp32$ git clone https://github.com/espressif/esptool
Clonage dans 'esptool'...
remote: Enumerating objects: 4622, done.
remote: Counting objects: 100% (1659/1659), done.
remote: Compressing objects: 100% (1000/1000), done.
remote: Total 4622 (delta 1125), reused 1124 (delta 638), pack-reused 2963
Réception d'objets: 100% (4622/4622), 11.54 Mio | 1.39 Mio/s, fait.
Résolution des deltas: 100% (2993/2993), fait.
tanguy@tanguy-Ubuntu:~/Bureau/tuto_esp32$ cd esptool/
tanguy@tanguy-Ubuntu:~/Bureau/tuto_esp32/esptool$ esptool.py --port /dev/ttyUSB0 erase_flash
esptool.py v3.2-dev
Serial port /dev/ttyUSB0
Connecting...
Detecting chip type... ESP32-C3
Chip is ESP32-C3 (revision 3)
Features: Wi-Fi
Crystal is 40MHz
MAC: 7c:df:a1:d0:15:28
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 16.5s
Hard resetting via RTS pin...
tanguy@tanguy-Ubuntu:~/Bureau/tuto_esp32/esptool$
```

Maintenant on va télécharger le firmware de l'esp32, rendez-vous à cette adresse :
<https://github.com/espressif/esp-at/releases/tag/v2.3.0.0> esp32c3

ESP32-C3 AT Release v2.3.0.0

Documentation for Release v2.3.0.0 is available at https://docs.espressif.com/projects/esp-at/en/release-v2.3.0.0_esp32c3/

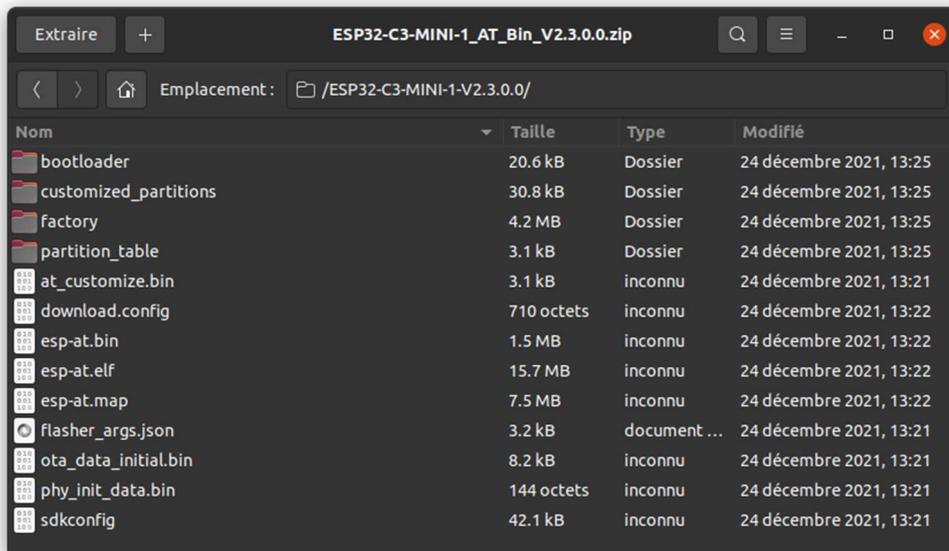
ESP32-C3 AT v2.3.0.0 is a major update for ESP32-C3 AT v2.2.0.0.

The firmwares:

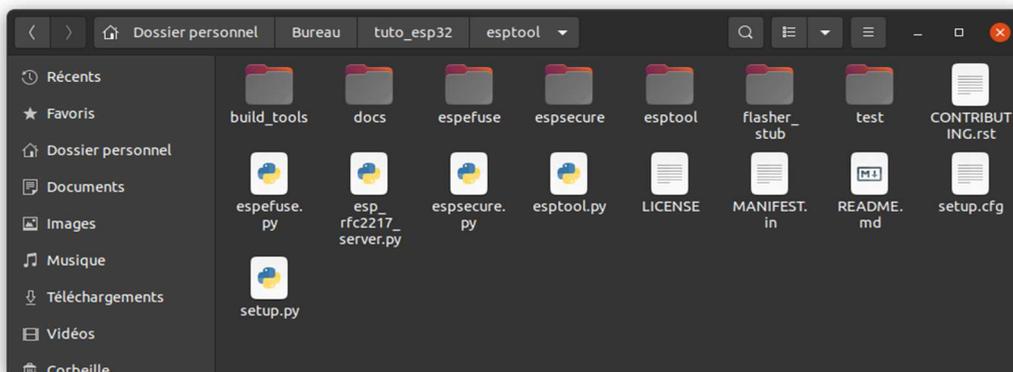
[ESP32-C3-MINI-1_AT_Bin_V2.3.0.0.zip](#)

Cliquez sur [ESP32-C3-MINI-1_AT_Bin_V2.3.0.0.zip](#)

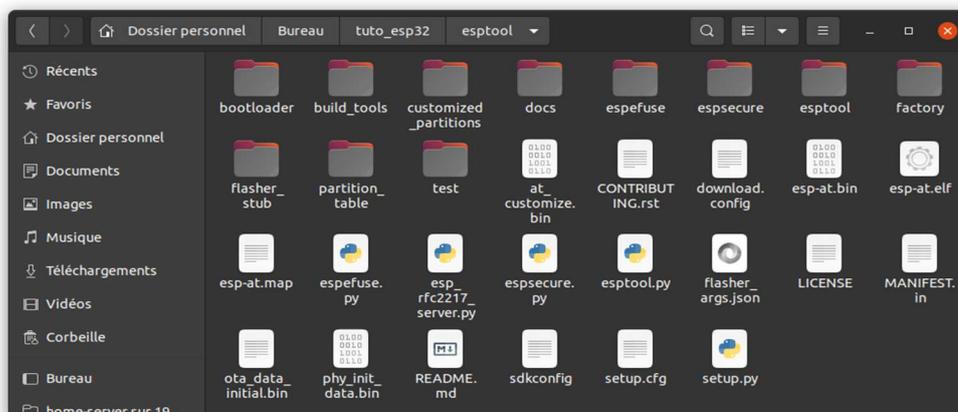
Ouvrez le zip, et naviguez dans le dossier ESP32-C3-MINI-1-V2.3.0.0 voici les fichiers qu'il contient :



Décompresser le contenu de ce dossier dans le dossier esptool.
Voici le dossier avant la manipulation :



et après :



Avant de continuer, éteignez et rallumez la carte.

Ensuite, lancez la commande :

```
> esptool.py --chip auto --port /dev/ttyUSB0 --baud 115200 --before default_reset --after hard_reset write_flash -z download.config
```

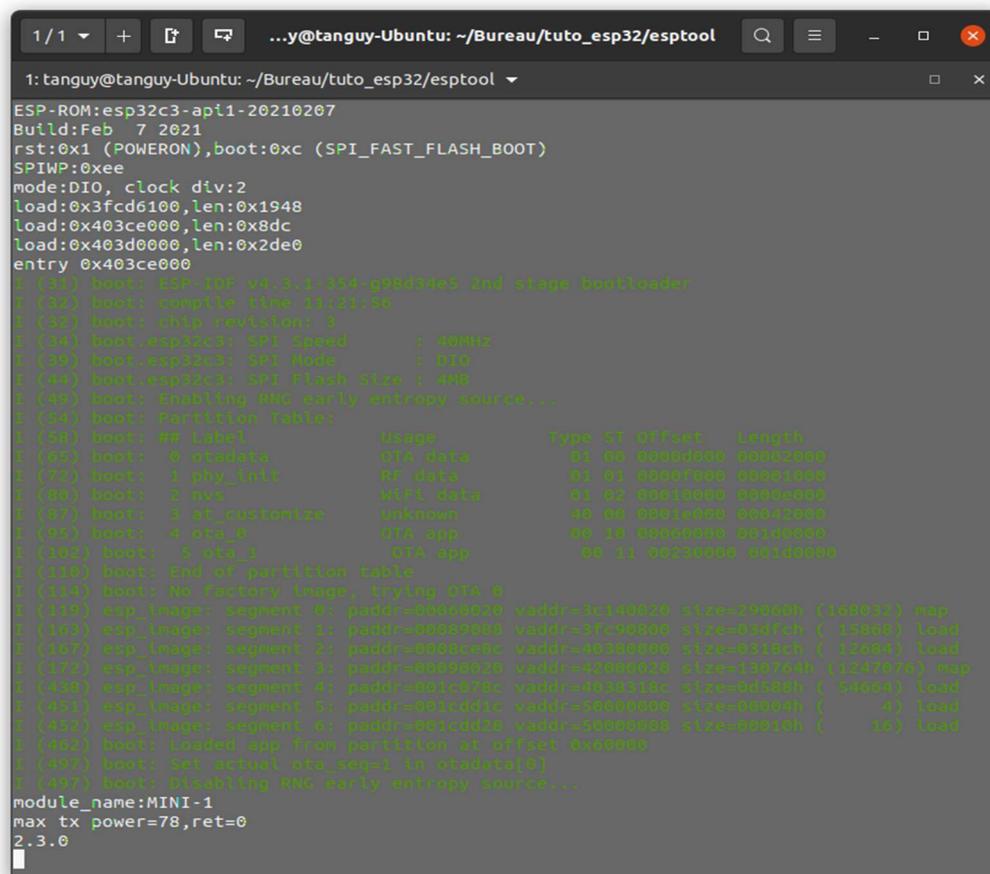
En remplacez download.config avec le contenu du fichier, dans mon cas :

```
> esptool.py --chip auto --port /dev/ttyUSB0 --baud 115200 --before default_reset --after hard_reset write_flash -z --flash_mode dio --flash_freq 40m --flash_size 4MB 0x8000 partition_table/partition-table.bin 0xd000 ota_data_initial.bin 0xf000 phy_init_data.bin 0x0 bootloader/bootloader.bin 0x60000 esp-at.bin 0x1e000 at_customize.bin 0x1f000 customized_partitions/ble_data.bin 0x3a000 customized_partitions/mqtt_key.bin 0x27000 customized_partitions/server_key.bin 0x3c000 customized_partitions/mqtt_ca.bin 0x2d000 customized_partitions/client_key.bin 0x2b000 customized_partitions/client_cert.bin 0x31000 customized_partitions/factory_param.bin 0x2f000 customized_partitions/client_ca.bin 0x38000 customized_partitions/mqtt_cert.bin 0x29000 customized_partitions/server_ca.bin 0x25000 customized_partitions/server_cert.bin
```

Normalement le programme devrait s'écrire dans la flash de l'ESP32.

Éteignez votre ESP32, puis connectez la pin IO8 et IO9 à 1 afin qu'il démarre en mode SPI (enlever le jumper).

Ensuite pour vérifier que tout s'est bien passé, ouvrez un picocom sur votre ESP32, puis rallumez la carte, voici ce que vous devriez obtenir :



```
1/1 + ...y@tanguy-Ubuntu: ~/Bureau/tuto_esp32/esptool
1: tanguy@tanguy-Ubuntu: ~/Bureau/tuto_esp32/esptool
ESP-ROM:esp32c3-api1-20210207
Build:Feb 7 2021
rst:0x1 (POWERON),boot:0xc (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:2
load:0x3fcd6100,len:0x1948
load:0x403ce000,len:0x8dc
load:0x403d0000,len:0x2de0
entry 0x403ce000
I (31) boot: ESP-IDF v4.3.1-354-g00d34e5 2nd stage bootloader
I (32) boot: compile time 11:21:54
I (33) boot: chip revision: 3
I (34) boot.esp32c3: SPI Speed      : 40MHz
I (35) boot.esp32c3: SPI Mode       : DIO
I (36) boot.esp32c3: SPI Flash Size : 4MB
I (40) boot: Enabling RNG early entropy source...
I (44) boot: Partition Table:
I (50) boot: ## Label                Usage            Type ST Offset   Length
I (55) boot: 0 atadata                OTA data         01 00 00000000 00002000
I (61) boot: 1 phy_init                 RF data          01 01 00007000 00001000
I (66) boot: 2 nvs                       WiFi data        01 02 00010000 0000e000
I (67) boot: 3 at_customize              unknown          00 00 0001e000 0001c000
I (73) boot: 4 ota_0                    OTA app          00 10 00000000 00100000
I (79) boot: 5 ota_1                    OTA app          00 11 00200000 00100000
I (84) boot: End of partition table
I (87) boot: No factory image, trying OTA 0
I (89) esp_image: segment 0: paddr=00000000 vaddr=3c100000 size=19800h (148800) map
I (103) esp_image: segment 1: paddr=00003000 vaddr=3f100000 size=030fch ( 12684) load
I (107) esp_image: segment 2: paddr=0000c000 vaddr=40300000 size=0331feh ( 13064) load
I (121) esp_image: segment 3: paddr=0000e000 vaddr=42000000 size=110760h (1147640) map
I (138) esp_image: segment 4: paddr=001c010c vaddr=403c0000 size=000000h ( 0) load
I (141) esp_image: segment 5: paddr=001cdd1c vaddr=50000000 size=000000h ( 0) load
I (142) esp_image: segment 6: paddr=001cd124 vaddr=50000000 size=000000h ( 0) load
I (162) boot: Loaded app from partition at offset 0x000000
I (166) boot: Set actual ota_seg1 to atadata[0]
I (167) boot: Disabling RNG early entropy source...
module_name:MINI-1
max tx power=78,ret=0
2.3.0
```

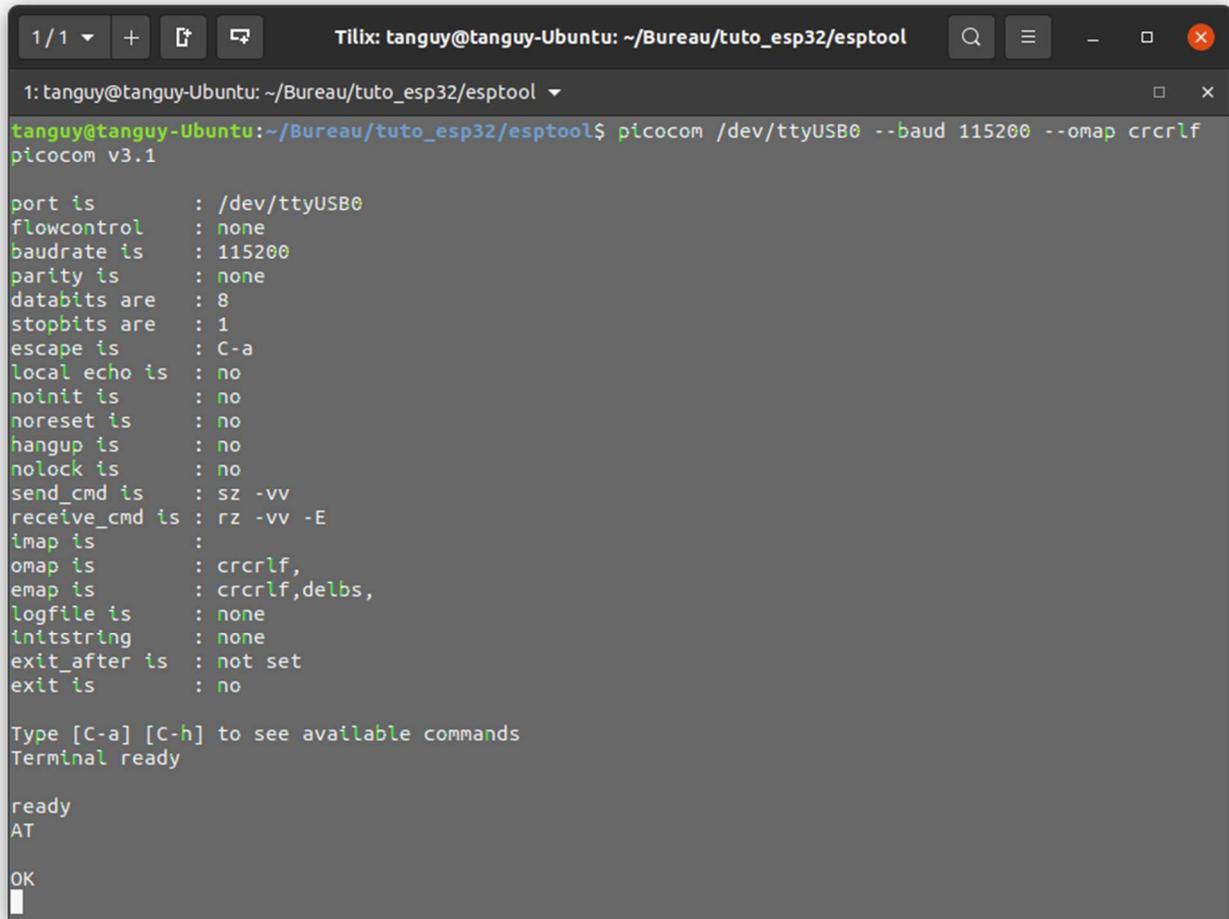
Si tout c'est bien passé, l'ESP32 démarre bien sur le firmware flashé, il ne reste plus qu'à changer d'UART pour communiquer avec les commandes AT.

Connecter votre convertisseur USB-UART au pin IO6 et IO7

Ouvrez un picocom avec cette commande :

```
> picocom /dev/ttyUSB0 --baud 115200 --omap crclrf
```

Éteignez et rallumez la carte, un ready devrait apparaître, entrez AT, puis faites ENTER, la carte devrait répondre OK



```
Tilix: tanguy@tanguy-Ubuntu: ~/Bureau/tuto_esp32/esptool
1: tanguy@tanguy-Ubuntu: ~/Bureau/tuto_esp32/esptool
tanguy@tanguy-Ubuntu:~/Bureau/tuto_esp32/esptool$ picocom /dev/ttyUSB0 --baud 115200 --omap crclrf
picocom v3.1

port is      : /dev/ttyUSB0
flowcontrol  : none
baudrate is  : 115200
parity is    : none
databits are : 8
stopbits are : 1
escape is    : C-a
local echo is : no
noinit is    : no
noreset is   : no
hangup is    : no
nolock is    : no
send_cmd is  : sz -vv
receive_cmd is : rz -vv -E
imap is      :
omap is      : crclrf,
emap is      : crclrf,delbs,
logfile is   : none
initstring   : none
exit_after is : not set
exit is      : no

Type [C-a] [C-h] to see available commands
Terminal ready

ready
AT
OK
█
```

La carte est prête à être programmée avec les commandes AT, bonne chance.