Realization of a connected cocktail machine with an espressif WROOM32



Revision	Data	Author	Modifications
R01	17/01/2018	Q.THEROND	Creation

1) Introduction	3
2) Sagittal diagramme	4
2.1) Description of the links	4
3) Functional diagram	5
3.1) FS1.0: Power supply	5
3.2) FS1.1: Stepper motors	5
3.3) FS1.2: RGB LED	6
3.4) FS1.3: uC wifi	6
3.5) FS1.4: Sensors	6
3.6) FS1.5: Pumps	6
3.7) FS1.6: Expansion connectors	7
4) Structural diagram	8
4.1) FS1.0: Power	9
4.1.1) Calculation of a radiator	10
4.2) FS1.1: Step motors	11
4.3) FS1.2: RGB LED	12
4.4) FS1.3 : uC wifi	13
4.4.1) Definition of the inputs, outputs of the microcontroller.	13
4.4.1.1) Inputs	13
4.4.1.2) Outputs	14
4.4.1.3) Other:	14
4.5) FS1.4: Sensors	15
4.6) FS1.5: Pumps	16
4.7) FS1.6: Expansion connectors	17
5) The PCB and List of components	18
5.1) PCB	18
5.2) PCB List of components	19
5.3) Meca and other:	20
6) The software	22
6.0) Presentation of Wifi/Bluetooth WROOM-32 module and SDK-IDF	22
6.0.1) WROOM-32 module	22
6.0.2) SDK-IDF	22
6.1) Flash partition	22
6.2) The architecture	24
6.2.1) The espressif SDK	24
6.2.2) The drivers	24
6.2.3) The HAL and OSAL	24
6.2.4) The application	24
6.3) The features of the software	25
	1

6.3.1) Wi-Fi connected mode	25
6.3.2) smartconfig mode	26
6.3.3) Update (OTA)	27
6.3.4) Push button	27
6.3.5) File system	28
6.3.6) End-of-run detection	29
6.3.7) Web page for the cmd cocktails	29
6.3.8) Motor control	31
6.3.9) Step motor control	31
6.3.10) RGB LED	32
6.3.11) cJson	32
6.3.12) mDNS hostname	33
6.3.13) Voice assistant (Alexa, Google Home,)	33
6.3.13.1) Configure NAT and dynDNS	33
6.3.13.2) Create an Applet on IFTTT	33
6.3.13.3) Open a TCP socket on the ESP32	34
6.3.14) Logs	35
6.3.15) Add-on	35
6.4) The JSON	36
6.4.1) The JSON of the location of the bottles	36
6.4.2) The JSON of the cocktail ingredient list.	36
6.5) The web page	37
6.5.1) The CSS	37
6.5.1) HTML	38
7) Mechanics	40
7.1) The wood	40
7.2) The tray	40
8) Getting started and functioning	41
8.1) Hardware connection	41
8.2) Configure the software	41
8.3) Compile and program the code	42
8.4) First boot	42
8.4) Movies	44
8.5) Hygiene of the cocktail machine	44
9) The future of the project	45
9.1) Modify	45
9.2) Add	45
10) Pictures	46

1) Introduction

Failing to achieve an interesting feature for the espressif SDK like wi-fi mesh I will present a fun project. The realization hardware and software of a connected cocktail machine with an espressif WROOM32. The goal is to be able to:

- Order a cocktail via a phone or PC
- Order a cocktail via a voice assistant (Google home, Alexa, ...).
- Order a cocktail via a IFTTT trigger

For that we will see different IOT functionality proposed by the WROOM32 like:

- Connect the system to internet with smartconfig
- Create a web page
- Use a mDNS and a fixed IP.
- Use a socket for IFTTT
- Update the system with OTA and the flash partition of ESP32
- Use the hardware features on the ESP (gpio, file system, ...)
- Use the FreeRTOS operating system of the espressif SDK-IDF
- Use the cJson library

And sorry for my English!

2) Sagittal diagramme



2.1) Description of the links

L2 / L2 ':

- Allows the user to order a cocktail via a web page.
- Allows the user to order a cocktail via a voice assistance (Alexa, Google Home, ...)
- Send his SSID and password with his smartphone to connect the system to the internet.
- Allows the system update with OTA
- L1:
 - Allows the user to put and retrieve his glass.
 - Lighting information about the state of the system.
 - Allows fill the bottles and clean them.



3.1) FS1.0: Power supply

The power function allows to create, thanks to an input voltage of 12V 5A, + 5V, + 3.3V, 0V.

Input(s): 12V, 5ADC.

Output(s): + 5V, + 3.3V, 0V

3.2) FS1.1: Stepper motors

The motor function allows to control the tray with the glass and to serve the cocktail via the dosers.

Input(s): G1: Digital signals (dir and clk) for controlling the stepper motors.

Output(s): Physical movement of the engines

3.3) FS1.2: RGB LED

The RGB LED informs the user of the system status.

Input(s): G2: Digital signals to control the RGB LED.

Output(s): L2: Lighting signals to inform the user

3.4) FS1.3: uC wifi

Ensures through a programmed treatment (software) the acquisition, processing and return of information. The smartconfig mode of the wroom32 allows to recover the SSID and password of the internet box. The wi-fi allows to control the system via a smartphone or a computer through a web page on a fixed IP address or with a voice assistance. It communicates with the motors, the RGB LED. sensors, and offers inputs / outputs to add additional functionality.

Input(s): L2 / L2 ': Wifi connection in wpa2, wep, ... smartconfig mode, http web page, socket for IFTTT, OTA update G4: Digital Signals for Limit Detection

Output(s):

L2 / L2 ': Wifi connection in wpa2, wep, ... smartconfig mode, http web page, socket for IFTTT, OTA update

G1: Digital Signals for Stepper Motor Control

G2: Digital Signals for RBG LED Control

G3: Digital signals for pump control

3.5) FS1.4: Sensors

Allows to delete the SSID and password of the wi-fi with a push button and the end-of-run detections.

Input(s): L1: Physical size

Output(s): G4: Digital signals on / off

3.6) FS1.5: Pumps

The "pumps" function is used to control DC motors.

Input(s): G3: Digital signals for pump control

Output(s): Physical movement.

3.7) FS1.6: Expansion connectors

Allows you to add features to the e-card.

Input(s): G5: Digital Signals

Output(s): G5: Digital Signals



4) Structural diagram

4.1) FS1.0: Power

The power function allows to create, thanks to an input voltage of 12V 5A, + 5V and + 3.3V.



J1: 12v input

C1, C11: Polarized chemical capacitors. They perform the filtering, and allow a decoupling in case of power supply

C2: Plastic capacitor, serves as an anti-parasite to suppress high frequencies (it is recommended in the technical documentation).

U3: 5V regulator, it allows to regulate the input voltage 12V DC in a voltage of 5V DC. According to the manufacturer's documentation of the 78XX, a minimum input voltage of the output voltage plus Vdrop (2V) or Ve = Vs + Vdrop = 5 + 2 = 7V minimum is required for proper operation.

U1: 3.3V LDO, it allows to regulate the input voltage 12V DC in a voltage of 3.3V DC.

P2, P4, P5: Choose the power source (internal or external) according to the current requirement. It is physically, connecting strips on which we just connect a "jumper" to select the desired voltage.

P3: Connector for testing output voltages.

P6: Connect an external power supply to not use the regulators (if more power is needed). Attention P2, P4, P5 must be connected correctly.

```
4.1.1) Calculation of a radiator
```

supposition: esp32=90mA, autre 50mA max Isystem = 90+50 = 140mA Pmax = (Tj - Ta)/RTHja = (125-25)/65 = 1,5WPutil(3.3V) = Isystéme*(Ve-Vs) = 140*10^-3*(12-3.3) = 1,218W Putil(5V) = Useless Putil<Pmax.

The utility of a radiator for regulators is not essential at room temperature.

4.2) FS1.1: Step motors

The motor function allows to control the tray with the glass and to serve the cocktail via the dosers.





IC1, IC2: are modules based on A4988 for the control of stepper motors powered by 3.3v. MSI_x inputs are used to configure the step resolution. the STEP input is a clock that triggers one step per clock period. the input DIR makes it possible to choose the direction of rotation of the motor.

P16, P17, P19, P20, P22, P32: Allows you to select one of the five state resolutions according to the truth table above. For the project I am in full step.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

C9, C10: Polarized chemical decoupling capacitors.

P18: Used to connect the motor of the tray with the glass (X axis).

P21: Used to connect the motors to serve the cocktail via the dosers (Y axis).

4.3) FS1.2: RGB LED

The RGB LED informs the user of the system status.



U4: Controlled switches (transistor network - ULN2803). Each of these 8 channels can be open or closed. It is capable of driving a load up to 500mA.

P1: Connect the RGB LED. PIN 5 is not used. You have to connect the LED as follow



(source: forum.arduino.cc/index.php?topic=465854.0)

4.4) FS1.3 : uC wifi

Ensures thanks to a programmed treatment (software) the acquisition the treatment and the return of the information. The smartconfig mode of WROOM32 allows to recover the SSID and password of the internet box. The wifi allows to control the system via a smartphone or a computer through a web page on a fixed IP address or with a voice assistance. It communicates with the motors, the RGB LED. sensors, and offers inputs / outputs to add additional functionality.



- R1, C5, P10: Allows a reset on the wroom32.
- C4, C3: used to filter high and low frequencies.
- P7: Allows programming the wroom32 via a UART.
- P12: Set wroom32 to programming mode.

4.4.1) Definition of the inputs, outputs of the microcontroller.

4.4.1.1) Inputs

Switch_MODE: Allows you to delete the Wi-Fi configuration (SSID and password)

Switch_GLASS_DETECTION: Detects the end of the board.

Switch_STOP_DETECTION: Used to detect the end of the dispenser management system.

Boot: Set the WROOM32 to programming mode.

Reset: Allows a reset on the WROOM32.

ESP32-Rx: Allows to send the binaries to be saved in memory of the wroom32

4.4.1.2) Outputs

ESP32_GPIO14, 12, 13, 15: Used to control the direction of rotation of motors and the speed of steps.

ESP32_GPIO23: Allows you to enable and disable A4988-based modules

LED_RED: Used to control the red LED. When the system starts, the LED is red as long as the system has not reached the limit

LED_GREEN: Used to control the green LED. Not used yet.

LED_BLEU: Used to control the BLUE LED. The blue fix indicates that the system is ready for use. The blinking blue indicates that the system is in preparation for a cocktail party.

ESP32-Tx: Allows you to send information to the programming tool.

PUMP_1 to PUMP_3: Used to control pumps.

4.4.1.3) Other:

SPARE1 to SPARE6: The SPARE inputs / outputs allow you to add new features to the cocktail machine such as milk powder dosing (of course, this implies a whole system of milk preservation and hygiene. given as an example.)

4.5) FS1.4: Sensors

Allows to delete the SSID and password of the wi-fi with a push button and the end-of-run detections.



R2, R3, R4: Pull-up resistance.

- C6, C7, C8: Anti-rebound capacitor.
- P9, P8: Used to connect the limit switches.

4.6) FS1.5: Pumps

The pump function is used to control DC motors.



P13, P14, P15: Used to connect the pumps.

U4: Switches controlled. Each of these 8 channels can be open or closed. It is capable of driving a load up to 500mA. After the tests the chosen pump consumes more than 500mA, so I added an expansion board with relays. You can connect your relays on P13, P14, P15 as follow.



(source: www.brunwinkel.de/elektronik/ics/uln2803/)

4.7) FS1.6: Expansion connectors

Allows you to add features to the board.



P11: Used to connect another electronic board to the central unit.

5) The PCB and List of components

5.1) PCB

The PCB does not have any particular constraint except to respect the "hardware design" proposed by expressif to have the best radio performances. *"make sure that the module is not covered by any metal shell. The antenna area of the module and the area 15 mm outside the antenna should be kept clean".*

The plans of the masses were put in transparency here for a better representation.



5.2) PCB List of components

Comment	Description	Designator	Footprint	LibRef	Quantity
EEEFK0J470UR	CAP 47uF 6.3V ALU CMS	C1, C9, C10	CAPA CHIMIQUE CMS	EEEFK0J470UR	3
GRM155R60J105KE1 9D	CAP 1UF 6V3 X5R CER 0402 CMS	C2	0402	GRM155R60J105KE1 9D	1
0805YD106KAT2A	CAP 10uF 16V X5R CER 0805 CMS	C3, C11	0805 POLARISE	0805YD106KAT2A	2
CC0402KRX7R6BB10 4	CAP 100NF K10V X7R CER 0402 CMS	C4	0402	CC0402KRX7R6BB10 4	1
MC0603B104K250CT	CAP 100NF 25V X7R CER 0603 CMS	C5, C6, C7, C8	0603	MC0603B104K250CT	4
A4988	MOTOR DRIVER	IC1, IC2	DRIVER MOTEUR	A4988	2
LD-0223-T25	CMS JACK SOCKET PLUG	J1	PJACK-LIHSHE NG-LD-0223-25	LD-0223-T25	1
HEADER 1x5	EMBASE MALE 1 RANGEE VERT 5 VOIES	P1	EMBASE 5 VOIES	HEADER 1x5	1
HEADER 1x3	EMBASE MALE 1 RANGEE VERT 3VOIES	P2, P4, P5, P7, P16, P17, P19, P20, P22, P23	EMBASE 3 VOIES	HEADER 1x3	10
HEADER 1x4	EMBASE MALE 1 RANGEE VERT 4 VOIES	P3, P6, P18	EMBASE 4 VOIES	HEADER 1x4	3
HEADER 1x2	EMBASE MALE 1 RANGEE VERT 2 VOIES 2.54mm	P8, P9, P10, P12, P13, P14, P15	EMBASE 2 VOIES - 2.54mm	HEADER 1x2 - 2.54 mm	7
HEADER 1x6	EMBASE MALE 1 RANGEE VERT 6 VOIES	P11	EMBASE 6 VOIES - VERTICAL	HEADER 1x6	1
HEADER 1x8	EMBASE MALE 1 RANGEE VERT 8 VOIES	P21	EMBASE 8 VOIES - VERTICAL	HEADER 1x8	1

MC 0.0625W 0402 1% 10K	RESIST.CMS 0402 1/16W 5% 10K	R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17	0402	MC 0.0625W 0402 1% 10K	17
Switch	SWITCH TACTILE	S1	TACT SWITCH 6X6	DTSM-61R-V-*	1
LM3940IMP-3.3/NOP B	REGULATEUR LINEAIRE 3V3	U1	SOT-223	LM3940IMP-3.3/NOP B	1
ESP-WROOM-32	MODULE WIFI ESPRESSIF WROOM 32	U2	MODULE_WIFI ESPRESSIF WROOM 32	ESP WROOM 32	1
LM78L05ACZ/NOPB	REGULATEUR LINEAIRE 5V	U3	TO-92	LM78L05ACZ/NOPB	1
ULN2803A	Transistor en réseau bipolaire	U4	DIP 18	ULN2803A	1

5.3) Meca and other:

BQLZR 500mm Longueur 8mm Diam […] tre ext […] rieur Axe linéaire horizontal Axe optique Ensemble de palier […] ¤ glissi […] re et axe linéaire Ensemble de soutien	https://www.amazon.fr/gp/product/B01K4M X1WW/ref=oh_aui_detailpage_o03_s00?ie =UTF8&psc=1
Myarmor 2 pcs 2 GT 20 dents Timing	https://www.amazon.fr/gp/product/B01K4M
Poulie de roue + 5 m 2 Gt-6 mm Ouverture	X1WW/ref=oh_aui_detailpage_o03_s00?ie
en caoutchouc Ceinture	=UTF8&psc=1
XCSOURCE NEMA 17 2 Phase Moteur	https://www.amazon.fr/gp/product/B011NR
4-Wire 1.8A Stepper 42 * 42 * 34mm Pour	MXYO/ref=oh_aui_detailpage_o02_s00?ie=
Imprimante 3D TE225	UTF8&psc=1
Chenxi Shop 15 x 15 mm L1000 mm	https://www.amazon.fr/gp/product/B073XF
Plastique câble Drag Fil de chaîne de	G975/ref=oh_aui_detailpage_o03_s00?ie=
transport pour CNC Router machine	UTF8&psc=1
Doseur d'alcool Beaumont 25ml	https://www.amazon.fr/gp/product/B009VJ4 M32/ref=oh_aui_detailpage_o04_s00?ie=U TF8&psc=1
SODIAL(R) 10 Pcs Mini Micro Fin de	https://www.amazon.fr/gp/product/B00U8M
course Levier a galet bras SPDT Declic	PR8U/ref=oh_aui_detailpage_o07_s00?ie=
LOT	UTF8&psc=1

Transformateur 220Vac vers 12Vdc Chargeur Alimentation à découpage pour guirlande Ruban Led Bande 5m 60W	https://www.amazon.fr/gp/product/B06WGR XCPK/ref=oh_aui_detailpage_o05_s00?ie= UTF8&psc=1
TecTake Boissons cuillère pour 6 bouteilles pour le mur bouteille support bar butler	https://www.amazon.fr/gp/product/B0196P9 XR0/ref=oh_aui_detailpage_o07_s00?ie=U TF8&psc=1
Anycubic 20 Dents Alumium Timing Poulie 5mm Alésage Ceinture Roue D'entraînement pour 10mm Largeur GT2 Ceinture 5PCS	https://www.amazon.fr/gp/product/B06XT27 4LH/ref=oh_aui_detailpage_o08_s00?ie=U TF8&psc=1
Popprint 5 pcs Ramps1.4 A4988 pilote de moteur pas à pas avec dissipateur thermique pour imprimante 3d, Green, 5	https://www.amazon.fr/gp/product/B06Y28H 956/ref=oh_aui_detailpage_o00_s00?ie=U TF8&psc=1
Anself Ultra silencieux Micro Mini DC12V Pompe à eau	https://www.amazon.fr/gp/product/B00OIHL TME/ref=oh_aui_detailpage_o03_s01?ie=U TF8&psc=1

6) The software

It is available with this document in zip file and on github <u>https://github.com/Mras2an/cocktail-machine</u>

6.0) Presentation of Wifi/Bluetooth WROOM-32 module and SDK-IDF

6.0.1) WROOM-32 module

I chose the Wi-fi/Bleutooth WROOM-32 module because this module has many advantages. Like an attractive price (for a wi-fi module), a nice memory capacity, a simple and complete SDK with lot of example and a stable wi-fi connection. So it's a great choice for making a homemade connected cocktail machine. But like all products, it also has drawbacks. The disadvantages of the resident module is in the flash protection management which requires 30 seconds (at first boot) to encrypt the 4Mb. It also lacks real hardware acceleration for ssl handshake (between 1 and 5 seconds for RSA256 in 2048).

6.0.2) SDK-IDF

I chose SDK-IDF on 2.1 branch instead of Arduino SDK because I already know the Arduino SDK. The SDK-IDF has a lot of contributors and I was able to propose three patches for the SDK-IDF. My pull requests was integrated in cind days, so the project maintainers are responsive.

6.1) Flash partition

To allow the update of the system via the wifi we need more partition. Espressif proposes the possibility of creating a cvs file to define the addresses. Here is the one of the cocktail machine:

Name, Type, SubType, Offset, Size, Flags otadata,data,ota,0xd000,8K, phy_init,data,phy,0xf000,4K, factory,app,factory,0x10000,1M, ota_0,app,ota_0,0x110000,1M, ota_1,app,ota_1,0x210000,1M, nvs,data,nvs,0x315000,500K, So we have a partitioned memory as follows:

I	Addr	Binaries
	0x001000	Bootloader.bin
li	0x008000	Partitions.bin
İ	0x010000	Factory.bin
	0x110000	OTA_0.bin
	0x210000	OTA_1.bin
	0x315000	NVS (500Ko)
	0x3F0000	Free

6.2) The architecture

The software is architectured as follows:



6.2.1) The espressif SDK

The SDK used is the SDK-IDF on the v2.1 branch available on github at (https://github.com/espressif/esp-idf). This is the official development system of the ESP32 chip.

6.2.2) The drivers

Used for different hardware interactions such as motors, the RGB LED, or the functionality of the WROOM32 module (Gpio, wifi, smartconfig, OTA, ...).

6.2.3) The HAL and OSAL

HAL ans OSAL allows simple porting of cocktail machines to another SDK, OS, or hardware platform.

6.2.4) The application

The application proposes the web page and the management of cocktails, the OTA update and a socket to read IFTTT command.

6.3) The features of the software

Connected cocktail machine software functionalities						
WIFI connection	Smartconfig	Update OTA	Push button			
Logs	File system	End-of-run detection	Web page			
IFTTT socket	mDNS	Motor commands	Step Motor commands			
RBG Led command	Reader Json	Add-on				

6.3.1) Wi-Fi connected mode

Connection to an internet box with WEP, WPA-PSK [TKIP], WPA2-PSK [AES] or WPA-PSK [TKIP] + WPA2-PSK [AES] security.

The function Esp32wifi_init allows you to set the wi-fi for your country and fill the structure "wifi_config_t" with your SSID and password.

```
wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
ESP_ERROR_CHECK(esp_wifi_init(&cfg));
ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));
ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
wifi_config_t sta_config = Esp32Wifi_getSSIDAndPass();
ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &sta_config));
ESP_ERROR_CHECK(esp_wifi_set_country(WIFI_COUNTRY_EU));
ESP_ERROR_CHECK(esp_wifi_start());
```

The Esp32_init function is used to configure an ip static on the wroom32. If your network is not 192.168.1.x with a mask in 255.255.255.0 you must modify the code below.

tcpip_adapter_init(); ESP_ERROR_CHECK(tcpip_adapter_dhcpc_stop(TCPIP_ADAPTER_IF_STA)); tcpip_adapter_ip_info_t info = { 0,}; IP4_ADDR(&info.ip, 192, 168, 1, 51); IP4_ADDR(&info.gw, 192, 168, 1, 1); IP4_ADDR(&info.netmask, 255, 255, 255, 0); ESP_ERROR_CHECK(tcpip_adapter_set_ip_info(TCPIP_ADAPTER_IF_STA, &info))

6.3.2) smartconfig mode

Allows sending the SSID and password in wi-fi broadcast. When the system has finished smartconfig the task Esp32SmartConfig_task saves the information in memory.

```
if((this->pass != NULL) && (this->ssid != NULL))
{
    Wifi_saveSSIDAndPass(this->ssid, this->pass);
    OsFree(this->ssid);
    OsFree(this->pass);
}
```

If the backup file exists then the smartconfig task is not enable to boot.

```
void Esp32SmartConfig init(void)
{
 this->smartConfigEnable = 0;
  char * ssid = (char *)Wifi_getSsid();
  char * pass = (char *)Wifi getPassword();
  if((ssid == NULL) || (pass == NULL))
  {
        this->smartConfigEnable = 1;
        wifi event group = xEventGroupCreate();
        this->ssid = NULL;
        this->pass = NULL;
 }
  if(ssid != NULL)
        OsFree(ssid);
  if(pass != NULL)
        OsFree(pass);
}
 void Esp32SmartConfig_createTask()
{
 if(this->smartConfigEnable)
        OsTaskCreate(Esp32SmartConfig task, "Esp32SmartConfig task", 4096, NULL,
 3, NULL);
}
You can see Esp32SmartConfig.c for more informations.
```

I choose the smatconfig mode instead of blufi (bluetooth wifi config connection to ap) because there is no example of smartphone application for los. While there are Andoid and los example apps for smartconfig mode.

6.3.3) Update (OTA)

Allows updating via wifi. To start the OTA functionality I used the cocktail machine web page. When you click on the Update button, the system reboot and start the update task.

```
char * update = Fs_read("Update", "Update");
if (update != NULL)
{
    BarDebug_info("LED GREEN\n");
    LedRGBHandling_ExecuteLedTaskFromISR(GREEN_LED);
    Fs_delete("Update", "Update");
    OsFree(update);
    Ota_InitTask();
}
```

I have integrate the example of "classycodeoss" for the update function. We need of 3 functions, Esp32Ota_begin, Esp32Ota_writeHexData, and Esp32Ota_end. This functions use espressif SDK OTA functions: esp_ota_begin, esp_ota_write, esp_ota_get_boot_partition, esp_ota_end. When the connected cocktail machine boot you can see in the log the current partition (Factory, OTA_0, OTA_1). this address partition is laid out in the customPart file seen above.

```
void Esp32Ota_init()
{
    const esp_partition_t * currentBootPartition = esp_ota_get_boot_partition();
    memset(this->currentPartition, '\0', PART_LEN);
    memcpy(this->currentPartition, currentBootPartition->label,
    strlen(currentBootPartition->label));
    this->addr = currentBootPartition->address;
    BarDebug_info("Part: %s, addr:%08x \n", Esp32Ota_getCurrentPartition(),
    Esp32Ota_getCurrentBootAddr());
}
```

6.3.4) Push button

Allows you to delete the SSID and wifi password. The button is managed by an OS task (Idel, rising edge and falling edge).

```
typedef enum eButtonMode_t
{
IDLE_BUTTON,
RISING_BUTTON,
FALLING_BUTTON
} eButtonMode t;
```

6.3.5) File system

```
It is an abstraction of the NVS espressif SDK functions
#define Fs_write
                      Esp32Fs_write
 esp err t Esp32Fs write(const char * filename, const char * data, const char * key)
{
 nvs_handle my_handle;
  esp_err_t err;
  err = nvs_open(filename, NVS_READWRITE, &my_handle);
  if(err != ESP OK)
  {
        BarDebug_err("Open: %x\n", err);
        return err:
 }
  err = nvs_set_str(my_handle, key, data);
  if(err != ESP OK && err != ESP ERR NVS NOT FOUND)
  {
        BarDebug err("Write: %x\n", err);
        nvs_close(my_handle);
        return err;
 }
  err = nvs_commit(my_handle);
  if(err != ESP OK)
  {
        BarDebug err("Commit: %x\n", err);
        nvs close(my handle);
        return err;
 }
  nvs_close(my_handle);
  return err;
 }
Lets you read or write data to a file system. Example for the SSID and password wi-fi.
 if(Fs_write(SSID_WIFI_FILE, ssid, SSID_WIFI_FILE) != ESP_OK)
{
        BarDebug err("Error to save SSID\n");
}
if(Fs_write(PASSWORD_WIFI_FILE, password, PASSWORD_WIFI_FILE) != ESP_OK)
{
        BarDebug err("Error to save password\n");
 }
```

6.3.6) End-of-run detection

Allows you to initialize the system using the "MotorHandling_setInitialPosition" function. This function is called at startup and at each cocktail request.

```
uint32_t bp1[1], bp2[1];
BarDebug_info("Set motor at the initial position on y...\n");
Gpio get(DETECTION AXE Y, bp2);
BAR ERROR CHECK(Gpio set(MOTOR AXE Y DIR, BAR LEVEL HIGH));
int end = MOTOR LIMIT;
while(bp2[0] && (end != 0))
{
     Gpio get(DETECTION AXE Y, bp2);
     BAR ERROR CHECK(Gpio set(MOTOR AXE Y CLK, BAR LEVEL LOW));
     CpuDelay ms(1);
     BAR_ERROR_CHECK(Gpio_set(MOTOR_AXE_Y_CLK, BAR_LEVEL_HIGH));
     CpuDelay ms(1);
     end--:
}
BarDebug info("Motor is now at the initial position on y.\n");
BarDebug info("Set motor at the initial position on x...\n");
Gpio get(DETECTION AXE X, bp1);
BAR ERROR CHECK(Gpio set(MOTOR AXE X DIR, BAR LEVEL HIGH));
while(bp1[0])
{
     Gpio get(DETECTION AXE X, bp1);
     BAR ERROR CHECK(Gpio set(MOTOR AXE X CLK, BAR LEVEL LOW));
     CpuDelay ms(1);
     BAR ERROR CHECK(Gpio set(MOTOR AXE X CLK, BAR LEVEL HIGH));
     CpuDelay_ms(1);
}
BarDebug info("Motor is now at the initial position on x.n");
```

6.3.7) Web page for the cmd cocktails

Allows you to order a cocktail. Once the system connected to the wi-fi network is called "Cocktail_init" and "QueueCocktail_init".

The "Cocktail" task will retrieve the JSON information to save it in RAM in the structures below:

```
typedef struct
{
    char name[MAX_NAME_SIZE];
    char note[MAX_NOTE_SIZE_FOR_BOTTLE];
    int position;
} sBottle;
typedef struct
```

{
 char name[MAX_NAME_SIZE];
 int measure;
} sIngredient;

typedef struct cocktail
{
 char name[MAX_NAME_SIZE];
 sIngredient ingredient[MAX_INGREDIENT];
} sCocktail;

sCocktail cocktail[MAX_COCKTAIL];
sBottle bottle[MAX_BOTTLE];

When the "Cocktail" task has finished retrieving the information we call "Html_init". Who will create the HTML web page.

html_indexBegin(http_index_hml); html_indexTitle(http_index_hml); html_tabBegin(http_index_hml); html_tabColumnBegin(http_index_hml, "Selection"); html_tabColumnMiddle(http_index_hml, "Ajouter"); html_tabColumnEnd(http_index_hml, "Fourni"); Cocktail_createHtmlCodeForCocktails(http_index_hml); html_tabEnd(http_index_hml);

The function "Cocktail_createHtmlCodeForCocktails" will sort the available ingredients according to the bottles on the cocktail machine. This allows you to tell the user which ingredients are unavailable.

for(i = 0; i < MAX_COCKTAIL; i++)
{
 for(j = 0; j < MAX_INGREDIENT; j++)
 {
 if(cocktail[i].ingredient[j].name[0] != '\0')
 {
 if(Cocktail_isBottleExiste(cocktail[i].ingredient[j].name))
 {
 ...
}</pre>

The "QueueCocktail_receivedTask" task is called when a user selects a cocktail on the web page. If we ask for three cocktails then they will be put on the waiting list. The variable "goToPosition" and "currentPosition" make it possible to calculate the future position of the tray according to the current state.

OsQueueReceive(pCtx->xQueueCocktailEventQueue, &QueueCocktail, OsPortTimingPeriod); LedRGBHandling_ExecuteLedTaskFromISR(BLUE_LED_FAST_BLINKING);

```
MotorHandling setInitialPosition();
int nblngredients = Cocktail getDispolngredients(bottleList.bottle, bottleList.position,
bottleList.measure, QueueCocktail);
int goToPosition = 0;
int currentPosition = 0;
for(int i = 0; i < nblngredients; i++)
{
       if(currentPosition != bottleList.position[i])
       {
               goToPosition = bottleList.position[i] - currentPosition;
               MotorHandling setPositionOnX(goToPosition);
               currentPosition += goToPosition;
               CpuDelay ms(500);
       }
       if(currentPosition != 0)
       {
               MotorHandling getAMeasureOnY(bottleList.measure[i]);
       }
       else
       {
               MotorHandling getAMeasureOnPump(bottleList.measure[i]);
       }
}
MotorHandling setInitialPosition();
LedRGBHandling ExecuteLedTaskFromISR(BLUE LED);
```

6.3.8) Motor control

Used to control a DC motor in position 0. For the moment only one motor is possible.

```
BAR_ERROR_CHECK(Gpio_set(MOTOR_PUMP_1, BAR_LEVEL_HIGH));
for(int j = 0; j < measure; j++)
{
        CpuDelay_ms(1000);
}
BAR_ERROR_CHECK(Gpio_set(MOTOR_PUMP_1, BAR_LEVEL_LOW));</pre>
```

6.3.9) Step motor control

Controls the X and Y axes and their senses. Below is an example for the X axis. The "MOTOR_OFFSET" allows to advance the plate of 10cm (distance between each bottle). "MOTOR_OFFSET" is calculated according to the degree of a step and the perimeter of the pulley.

```
if(position > 0)
{
     BAR_ERROR_CHECK(Gpio_set(MOTOR_AXE_X_DIR, BAR_LEVEL_LOW));
     end = position * MOTOR_OFFSET;
}
else
```

```
{
    BAR_ERROR_CHECK(Gpio_set(MOTOR_AXE_X_DIR, BAR_LEVEL_HIGH));
    end = (position * (-1)) * MOTOR_OFFSET;
}
for(int i = 0; i < end; i++)
{
    BAR_ERROR_CHECK(Gpio_set(MOTOR_AXE_X_CLK, BAR_LEVEL_LOW));
    CpuDelay_ms(1);
    BAR_ERROR_CHECK(Gpio_set(MOTOR_AXE_X_CLK, BAR_LEVEL_HIGH));
    CpuDelay_ms(1);
}</pre>
```

6.3.10) RGB LED

Used to control the color of the LED to inform the user. This is a task that manages the timing of the flashing. The variable "enableLed" allows to enable or disable the LED. The variable "queueReceiveDelay" is used to manage the flashing speed in milliseconds.

```
OsQueueReceive(tsQueueForLed, &eAsyncMsg, queueReceiveDelay);
eAsyncCurrent = eAsyncMsg;
if(IDEL_LED == eAsyncMsg)
{
    LedRGBGpioDriver_SetColor(LED_NOT_DEFINED);
}
else if((BLUE_LED_FAST_BLINKING | enableLed) == eAsyncMsg)
{
    queueReceiveDelay = BLINK_FAST;
    if(!LedRGBGpioDriver_ToggleColor(LED_BLUE))
    {
        BarDebug_info("LED NOT DEFINED");
    }
}
....
```

6.3.11) cJson

Allows to recover the fields of JSON thanks to the cJon library. Here is an example to retrieve information from a bottle.

```
cJSON * _name = cJSON_GetObjectItem(_bottle, "name");
if(_name != NULL)
{
     cJSON * _note = cJSON_GetObjectItem(_bottle, "note");
     if(_note != NULL)
     {
        cJSON * _position = cJSON_GetObjectItem(_bottle, "position");
        if(_position != NULL)
        {
            memcpy(bottle[i].name, _name->valuestring, strlen(_name->valuestring));
        }
    }
}
```

memcpy(bottle[i].note, _note->valuestring, strlen(_note->valuestring)); bottle[i].position = (int) _position->valuedouble;

•••

6.3.12) mDNS hostname

Allows access to the web page without the ip address. Example: <u>http://mybar.local/</u> instead of <u>http://192.168.1.51</u>. It works on my PC but not on my phone.

mdns_server_t * mdns = NULL; mdns_init(TCPIP_ADAPTER_IF_STA, &mdns); ESP_ERROR_CHECK(mdns_set_hostname(mdns, "mybar")); ESP_ERROR_CHECK(mdns_set_instance(mdns, "mybar")); ESP_ERROR_CHECK(mdns_service_add(mdns, "_http", "_tcp", 80)); ESP_ERROR_CHECK(mdns_service_instance_set(mdns, "_http", "_tcp", "mybar"));

You can also ping the cocktail machine with:

\$ ping mybar.local PING mybar.local (192.168.1.51) 56(84) bytes of data. 64 bytes from 192.168.1.51: icmp_seq=1 ttl=255 time=5.07 ms 64 bytes from 192.168.1.51: icmp_seq=2 ttl=255 time=2.76 ms

6.3.13) Voice assistant (Alexa, Google Home, ...)

To add a compatibility between the connected cocktail machine and a voice assistant we have lot of possibility. The most simple is to use IFTTT (If This Then That) to send a curl request to the ESP32 with "webhooks". For that we need to configure NAT and dynDNS on the internet box, create an Applet on IFTTT and open a TCP socket on the ESP32.

6.3.13.1) Configure NAT and dynDNS

we need to configure the NAT on our internet box to make a redirection on our public IP on our local IP. As our public IP is dynamic we use dynDNS as a DNS service to have a fixed domain name. All this configuration can be make on you internet box manager web page.

The domain name used is useful for the IFTTT trigger

6.3.13.2) Create an Applet on IFTTT

To create an applet on IFTTT with Google home, we have to create an account on "https://ifttt.com/". Click on "New applet". You will see (if "this" then "that"). Click on "this", select "Google assistant" and follow the instructions. After that click on "that", select "Webhooks", and follow the instructions. Webhooks, allows to send a "curl request" on the cocktail machine

URL	
http://YOURIP_OR_DNA	ME/cocktail
Surround any text with ">>" to content	o escape the
Method	
POST	-
The method of the request e.4 DELETE	g. GET, POST,
Content Type	
application/json	-
Optional	
Body	
{"name":"YOURCOCKTA	IL"}

In my example I use "Google home", but you can make the same trigger with "Alexa", "Button widget", "location", "date & time", and more. Example with "Button widget" you can create a favorite cocktail widget on your desk phone.



6.3.13.3) Open a TCP socket on the ESP32

To open a TCP socket on the ESP32 I use "socket", "bind", "accept" functions. I use "read" and "write" functions for data communications. when I receive the "request post" send by IFTTT, I check if the cocktail name is available in the cocktail list and I answer the HTTP status code "HTTP/1.1 204" or "HTTP/1.1 400".

```
static const char HeadPostHttp[] = "HTTP/1.1 204 No Content\r\n\r\n";
static const char HeadPostHttpErr[] = "HTTP/1.1 400 Bad Request\r\n\r\n";
...
cJSON * _name = cJSON_GetObjectItem(_root, "name");
if(_name != NULL)
{
    int numCocktail = Cocktail_isCocktailExiste(_name->valuestring);
```

if(numCocktail != 255)
{
 write(sock, HeadPostHttp, strlen(HeadPostHttp));
 QueueCocktail_received(numCocktail);
 }
 else
 {
 write(sock, HeadPostHttpErr, strlen(HeadPostHttpErr));
 }
}

6.3.14) Logs

I have two logs level defined by "BAR_DEBUG", info level and error level. If "BAR_DEBUG" is defined info and error logs is enable. If "BAR_DEBUG" is not defined only error level is available.

#ifdef BAR_DEBUG
#define BarDebug_info(msg, ...) OsPrintf("[INFO: %s, L%d] "msg, __FUNCTION__,
__LINE__, ##__VA_ARGS__)
#define BarDebug_err OsDebug_err
#else
#define BarDebug_info(msg, ...)
#define BarDebug_err OsDebug_err
#endif

6.3.15) Add-on

Nothing yet.

6.4) The JSON

there are two JSONs in the project. The first allows to define the location of the bottles and the second the list is ingredient of the cocktails.

6.4.1) The JSON of the location of the bottles

The "bottles" table below is a bottle list with a name and a position. Te careful the position must exist physically on the cocktail machine. Position 0 is the initial position of the tray.

```
{
   "bottles": [{
                  "bottle": {
                           "name": "eau",
                          "note": "0%/vol",
                           "position": 0
                 }
         }, {
                  "bottle": {
                           "name": "jus de pomme",
                          "note": "0%/vol",
                           "position": 1
                 }
         },
         . . . .
    }]
}
```

6.4.2) The JSON of the cocktail ingredient list.

The cocktail table below is a cocktail list. A cocktail must have a name and a table "ingredients" which is an ingredient list. Each ingredient has a name (which is not necessarily available in the cocktail machine) and a "measure" which is the amount of liquid to serve (1 = 2ml, 2 = 4ml, ...).

"ingredient": { "name": "jus de citron", "measure": 1 } }, { "ingredient": { "name": "sucre de canne", "measure": 1 } }, { "ingredient": { "name": "eau gazeuse", "measure": 2 } }, { "ingredient": { "name": "glace pilee", "measure": 1 } }] } }, { }] }

When displaying a cocktail on the web page you can see two columns. A column for the ingredients available in the machine and a column for the ingredients to be added manually.

6.5) The web page

Yes but why a web page and not a mobile application? The answer is simple, because it is compatible on all smartphones and PCs.

The web page is accessible at the IP address <u>http://mybar.local/</u> or http://192.168.1.51 of your network. This is HTML and CSS code generated by the C code of wroom32. When the module starts, the software cycles through the JSON "bottles" and "cocktails" to create a dynamic HTML table with three columns and N line(s). In column one, we have the CSS buttons with the name of the cocktail (available in the JSON). In column two we have the ingredient(s) to add manually (not available in bottle list). In column three we have the ingredient (s) available in the cocktail machine (available in bottle list).

6.5.1) The CSS

In the CSS we have buttons, the font, the background color ...

<style></style>

color: white; text-align: center; } р{ font-family: verdana; font-size: 20px; } body { background-color: lightblue; } table { border-collapse: collapse; } td { border: 1px solid black; } .button { width:85px; height:85px; background:#fafafa; box-shadow:2px 2px 8px #aaa; font:bold 13px Arial; border-radius:50%; color:#555; } </style>

6.5.1) HTML

In the first line of the HTML code is the title and a link to an image. This line allows, when we add the shortcut of the web page on a smartphone to have a nice shortcut with a name and logo.



The rest of the HTML code is the three-column table. Here is an example below:

Welcome, on Quentin's bar!

Please select your shooter or cocktail:

Selection	Ajouter	Fourni
Mojito	menthe verte, jus de citron, sucre de canne, eau gazeuse, glace pilee,	rhum blanc,
TiPunch	jus de citron,	rhum blanc, sirop de sucre,

7) Mechanics

7.1) The wood

I bought wood and metal brackets at a hardware store. You need a drill, a circular saw, and a screwdriver.

7.2) The tray

You can easily make yourself the mechanical tray, however to avoid spending too much time on the realization of this mechanical part I use the 3D plans propose by "DIY Machines" (https://www.thingiverse.com/thing: 2478890/zip).

License:

Robotic Bartender (https://www.thingiverse.com/thing:2478890) by DIY_Machines is licensed under the Creative Commons - Attribution license. http://creativecommons.org/licenses/by/3.0/

8) Getting started and functioning

You have to clone the git repository: https://github.com/Mras2an/cocktail-machine

8.1) Hardware connection



8.2) Configure the software

After soldering your card and building the mechanical part of the system you have to configure some parameters.

1 - Board.h

Used to define the GPIO mapping of the system. Example for the RGB LED:

#define LED_GPIO_RED	BAR_GPIO_NUM_32
#define LED_GPIO_GREEN	BAR_GPIO_NUM_33
#define LED_GPIO_BLUE	BAR_GPIO_NUM_25

1 - the motors

you have to modify the "#define" of Motor.c to calibrate the number of steps according to the mechanics of the cocktail machine.

2 - The JSON

You have to bring in your bottles and create your cocktail.

8.3) Compile and program the code

1 - In a GNU / Linux terminal do an export of the SDK-IDF (branch 2.1) and the toolchain.

```
$ export IDF_PATH=/YOUR_PATH/esp-idf
```

```
$ export PATH=$PATH:/YOUR_PATH/xtensa-esp32-elf/bin/
```

2 - In the project make a "make"

3 - Connect the card with the jumper Boot. Then use the espressif flashing software.

```
$ python /esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 115200 --before default_reset --after hard_reset write_flash -z --flash_mode dio --flash_freq 40m --flash_size detect 0x1000 bootloader.bin 0x10000 cocktail-machine.bin 0x8000 customPart.bin
```

4 - Now you can use OTA functionality to update your system (you have to be connected to your home wi-fi). For that go to the web page (<u>http://mybar.local/</u> or http://192.168.1.51) and click on the button "Update" at the end of the page (the LED is green). Go to utils file and execute the command follow:

\$ python update_firmware.py 192.168.1.51 cocktail-machine.bin



8.4) First boot

1 - At start-up the LED is red, the Y-axis searches for the end position, then the X-axis searches for the end position. Once the system is initialized the LED goes blue.

2 - When you start the system for the first time, connect the system to your Wi-Fi box. To do this download the application "ESP8266 SmartConfig" on your phone's awning. Connect your phone to your wifi, and launch the espressif application. You just have to enter your wifi password and click on "confirm". The module will save in flash your SSID and password. Once saved the smartconfig mode will be disabled at startup.

🕿 🗉 🏺		💎 🖹 💈 12:58		
ESP82	266 SmartC	onfig		
SSID · NAT WRTNODE				
Password :				
SSID is Hidden				
CONFIRM				
\bigtriangledown	0			

- 3 Go to http://mybar.local/ or http://192.168.1.51 and choose your cocktail.
- 4 Enjoy :-)

8.4) Movies

You can watch videos on my youtube channel: https://www.youtube.com/watch?v=C-QNyAYIRnY https://www.youtube.com/watch?v=Tx3ExhTUp6A&t https://www.youtube.com/watch?v=QJvKOMsrSR0

8.5) Hygiene of the cocktail machine

For good hygiene, remember to clean the machine after each bottle switch.

9) The future of the project

The version presented is the version 1. Many things remain to be modified and added

9.1) Modify

- Be careful, on my board (PCB) I switch the 5v in 12v on the ULN2803 but you can use the 5 volts if you want.

9.2) Add

- Detection of the glass.

10) Pictures







