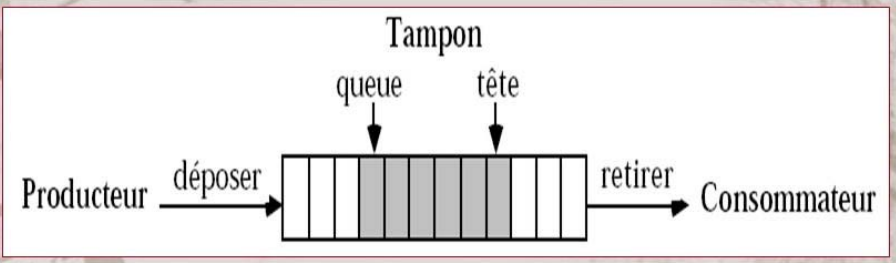
Module Systèmes Temps-Réels
TD FreeRTOS
sous ESP32-DevKitC v4

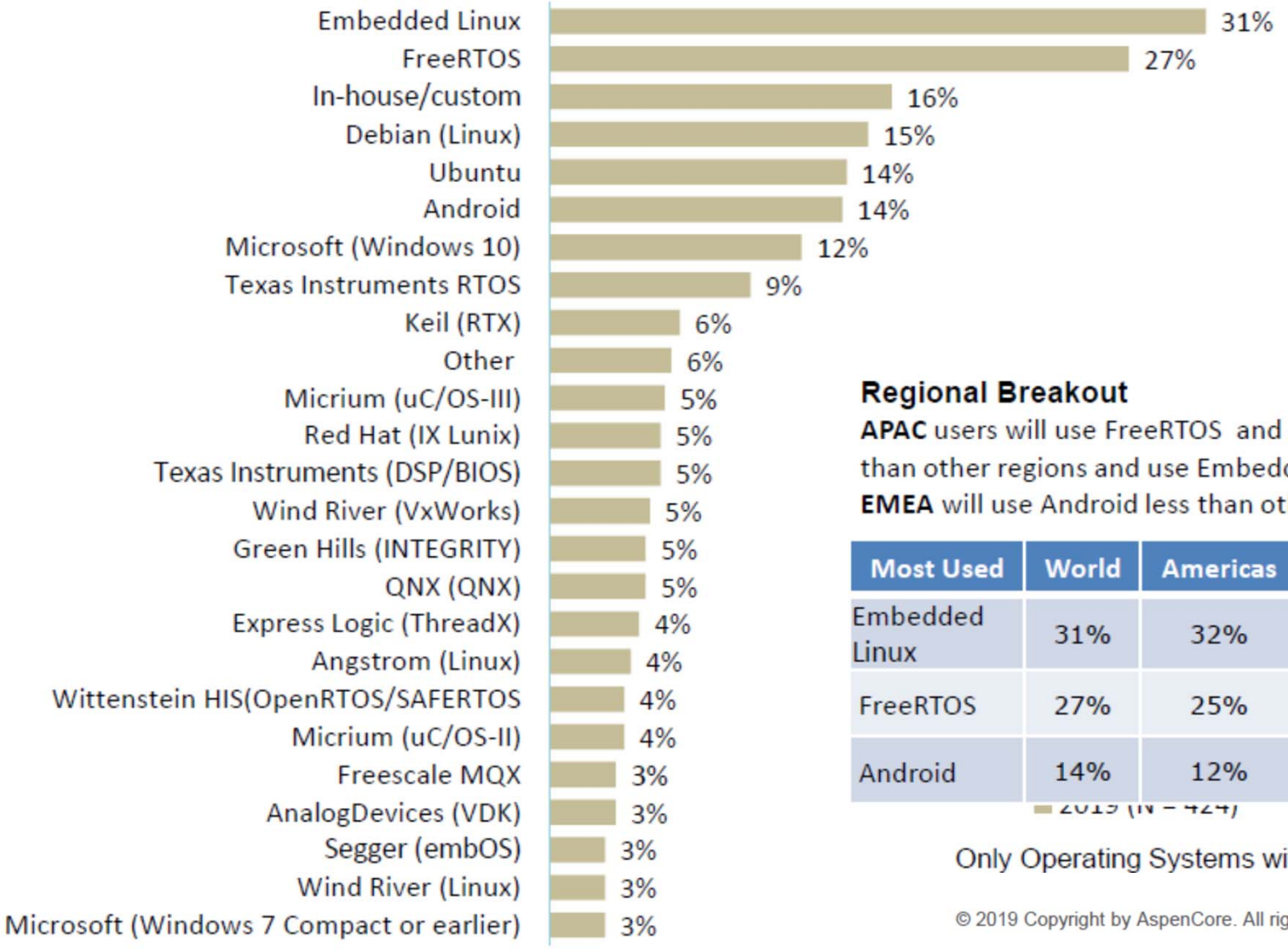
Modèle Producteur/Consommateur



- Les deux processus coopèrent en partageant un même tampon
  - Le producteur produit des objets qu'il dépose dans le tampon
  - Le consommateur retire des objets du tampon pour les consommer
- **Conflits**
  - Le producteur veut déposer un objet alors que le tampon est déjà plein;
  - Le consommateur veut retirer un objet du tampon alors que celui-ci est vide;
  - Le producteur et le consommateur ne doivent pas accéder simultanément au tampon.

FreeRTOS:

OS Temps-réel le plus utilisé au monde depuis 2011 dans le domaine de l'embarqué:



Regional Breakout

APAC users will use FreeRTOS and Android much more than other regions and use Embedded Linux much less.
EMEA will use Android less than other regions.

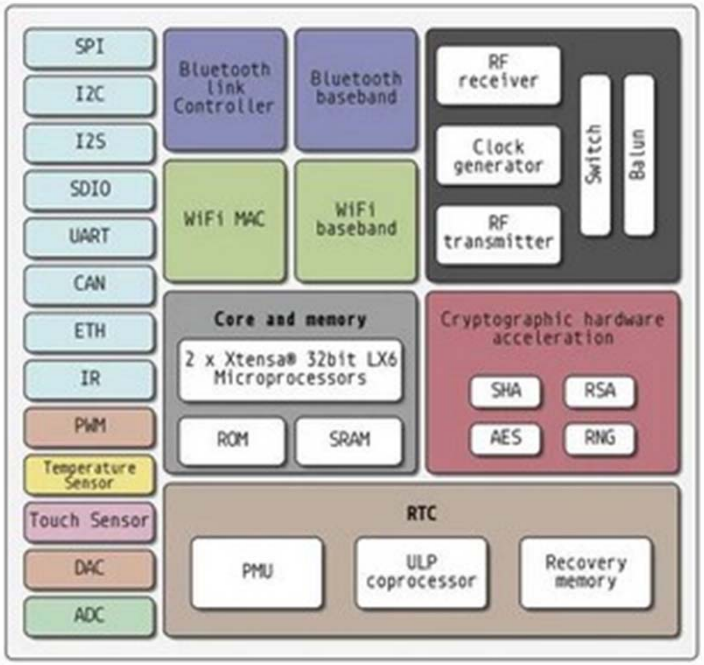
Most Used	World	Americas	EMEA	APAC
Embedded Linux	31%	32%	31%	**26%**
FreeRTOS	27%	25%	24%	**37%**
Android	14%	12%	**10%**	**26%**

2019 (N = 424)

Only Operating Systems with

Microcontrôleurs : ESP32-WROOM-32D

ESP32-DEVKITC-32D-F



Development Board Started With (Write-in recall answers only)	N=281	%
STMicroelectronics	43	15.3%
TI	30	10.7%
NXP	20	7.1%
Raspberry Pi	19	6.8%
Microchip	14	5.0%
Arduino	13	4.6%
Xilinx	13	4.6%
Atmel	11	3.9%
Espressif ESP-32	7	2.5%
Renesas	7	2.5%
Silicon Labs	6	2.1%
Nordic	5	1.8%
Digilent	4	1.4%
Nucleo Board	4	1.4%
ZedBoard	4	1.4%
Analog Devices	3	1.1%
Beaglebone Black	3	1.1%
Cypress	3	1.1%
AdaFruit 'Feather' Cortex-M4	2	0.7%
ARM	2	0.7%
Atmega	2	0.7%
Avnet Picozed	2	0.7%

- Proc. Xtensa 32 bits 2 cœurs à 240 Mhz
- 520 Ko SRAM / 448 Ko ROM
- Wifi, BLE, Ir
- 3 UART, 2 I^2C, 3 SPI, 4 timers
- 34 GPIO, 18 ADC, 2 DAC, 16 PWM

ESP32: Map Adress

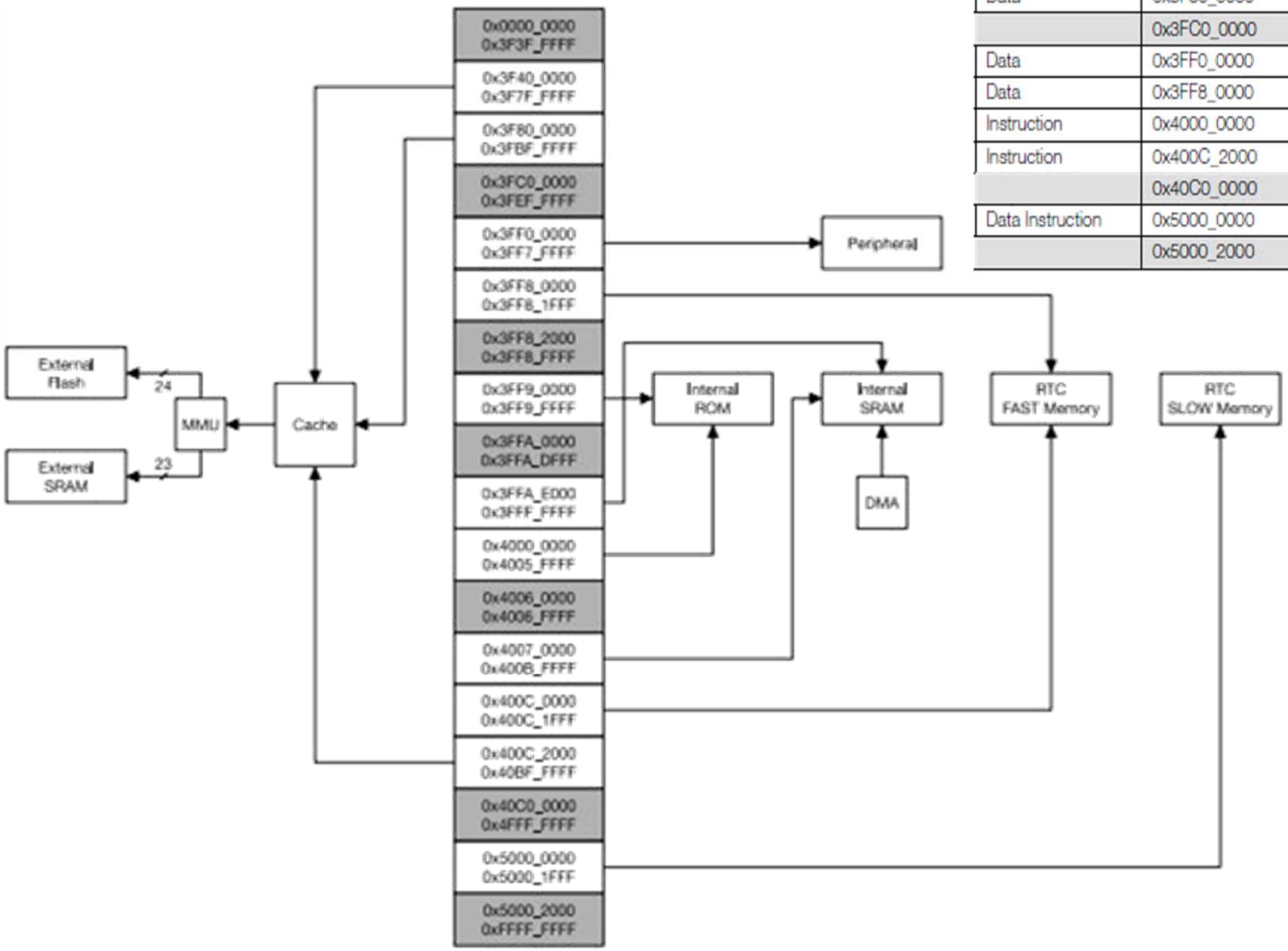
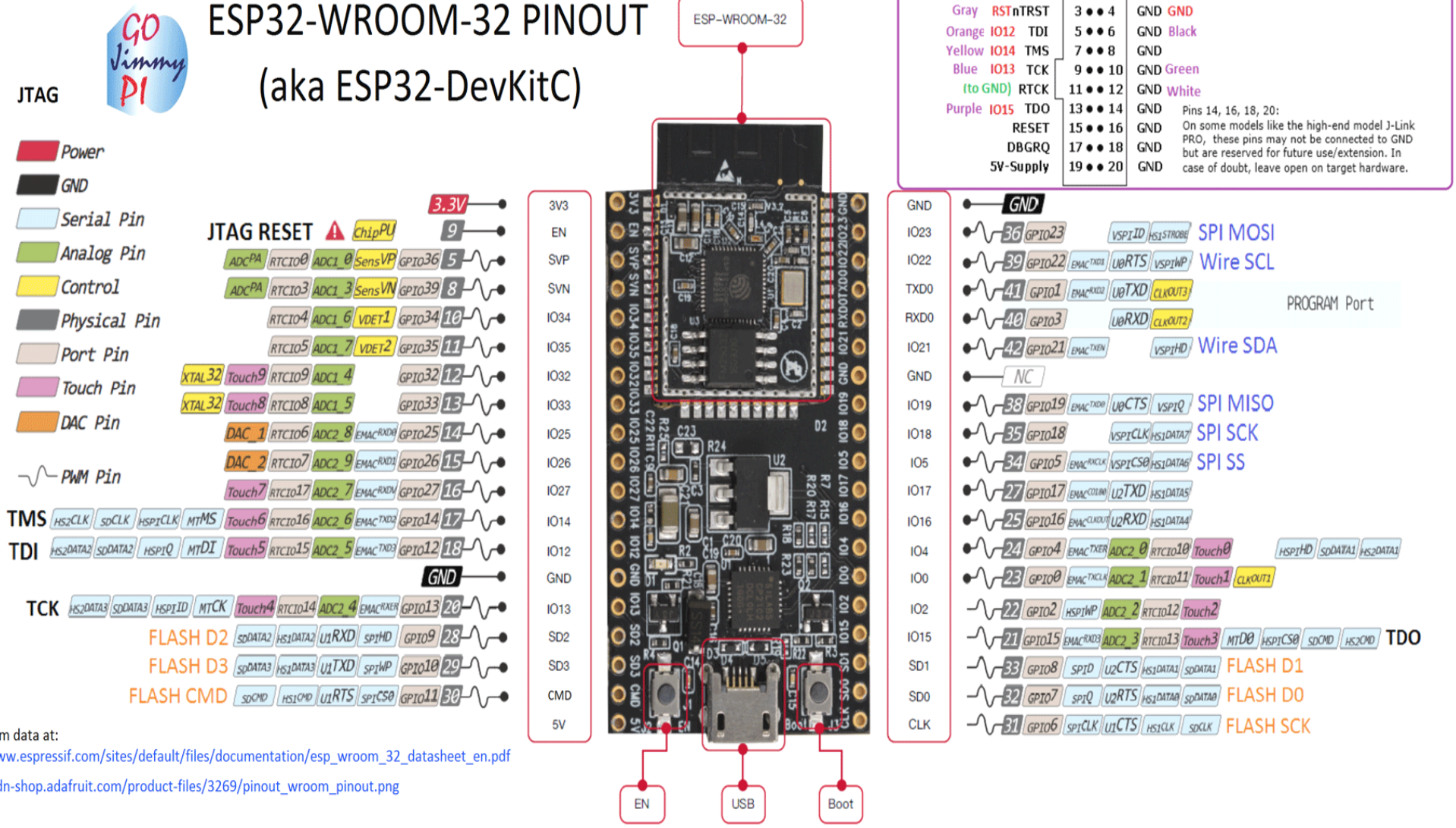


Table 1: Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
	0x0000_0000	0x3F3F_FFFF		Reserved
Data	0x3F40_0000	0x3F7F_FFFF	4 MB	External Memory
Data	0x3F80_0000	0x3FBF_FFFF	4 MB	External Memory
	0x3FC0_0000	0x3FEF_FFFF	3 MB	Reserved
Data	0x3FF0_0000	0x3FF7_FFFF	512 KB	Peripheral
Data	0x3FF8_0000	0x3FFF_FFFF	512 KB	Embedded Memory
Instruction	0x4000_0000	0x400C_1FFF	776 KB	Embedded Memory
Instruction	0x400C_2000	0x40BF_FFFF	11512 KB	External Memory
	0x40C0_0000	0x4FFF_FFFF	244 MB	Reserved
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	Embedded Memory
	0x5000_2000	0xFFFF_FFFF		Reserved

ESP32-WROOM-32 PINOUT
(aka ESP32-DevKitC)



JTAG 20-pin

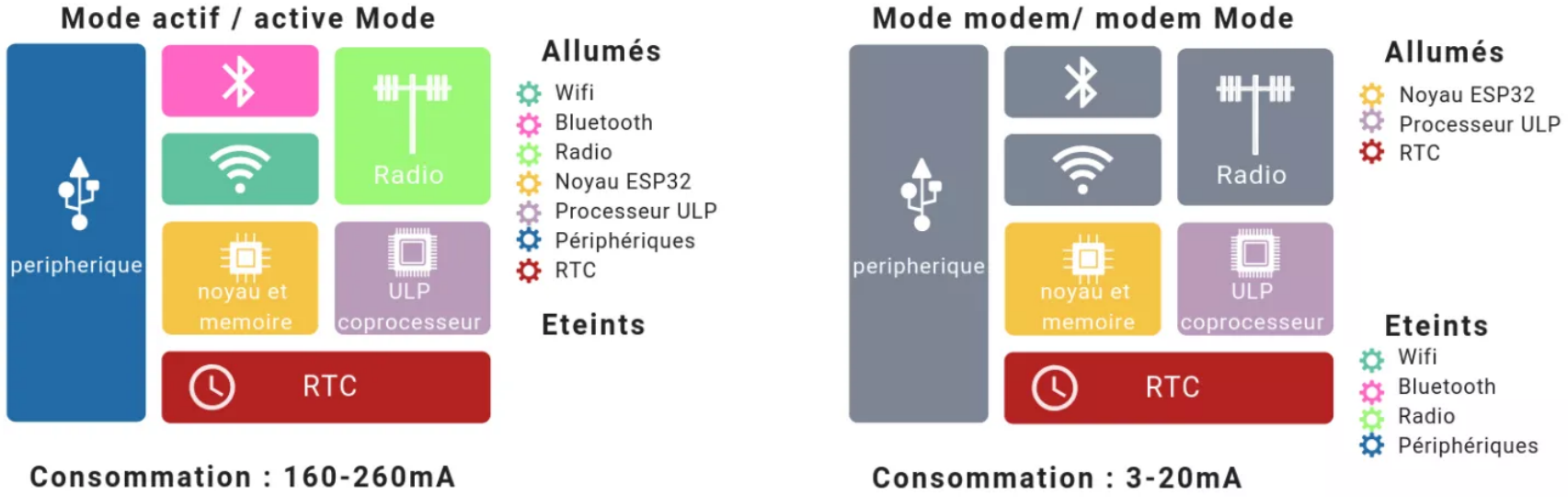
Red	3V3	VTref	1 • • 2	---	
Gray	RST	nTRST	3 • • 4	GND	GND
Orange	IO12	TDI	5 • • 6	GND	Black
Yellow	IO14	TMS	7 • • 8	GND	
Blue	IO13	TCK	9 • • 10	GND	Green
(to GND)		RTCK	11 • • 12	GND	White
Purple	IO15	TDO	13 • • 14	GND	Pins 14, 16, 18, 20:
		RESET	15 • • 16	GND	On some models like the high-end model J-Link
		DBGRQ	17 • • 18	GND	PRO, these pins may not be connected to GND
		5V-Supply	19 • • 20	GND	but are reserved for future use/extension. In case of doubt, leave open on target hardware.

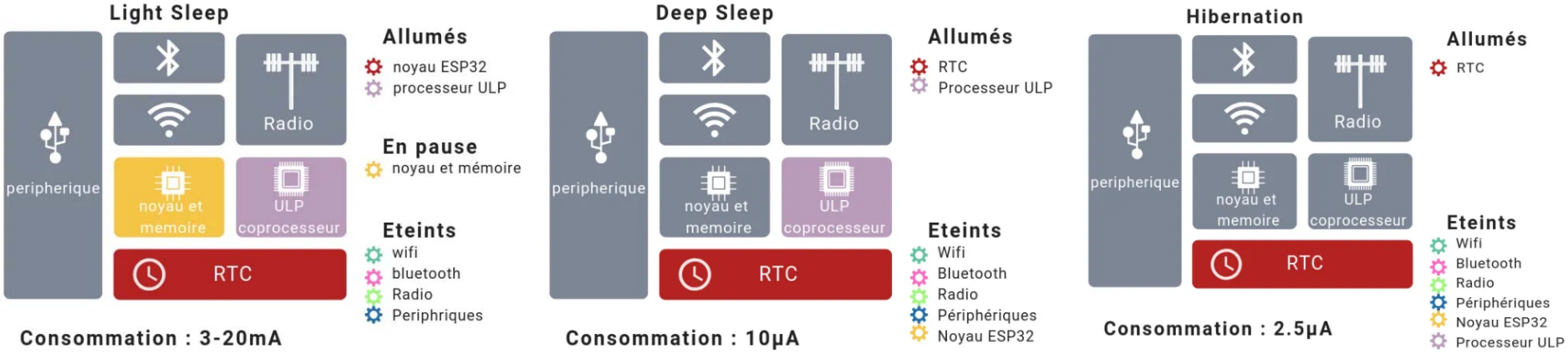
Adapted from data at:

http://www.espressif.com/sites/default/files/documentation/esp_wroom_32_datasheet_en.pdf

https://cdn-shop.adafruit.com/product-files/3269/pinout_wroom_pinout.png

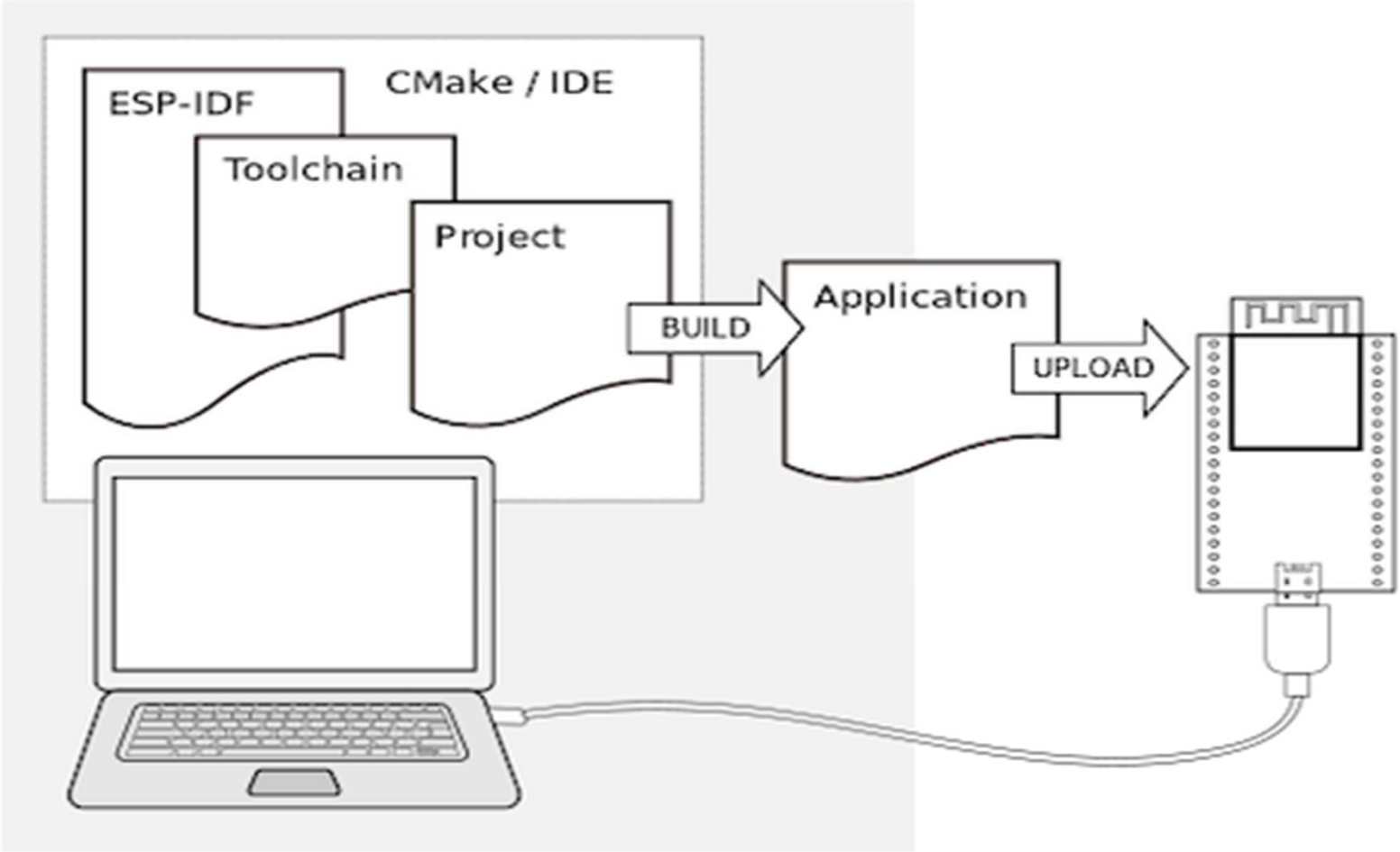
ESP32: 5 modes de fonctionnement





Environnement de développement:

ESP32-DevKit-C / PlateformIO / Visual Studio Code



Environnement de développement:

- On crée un projet avec le fichier **platformio.ini** suivant:
  - Avec le *Framework* Espressif (ESP-IDF):

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = espidf
monitor_speed = 115200
```

  - Avec le *Framework* Arduino:

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200
```

- Il faudra inclure les fichiers d'en-tête suivants:

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "sdkconfig.h"
```

```
#include <Arduino.h>
```

- Si l'ESP n'est pas détecté lors du déploiement, il faut installer le driver CP210x

Structure du projet: (http://tvaira.free.fr/esp32/esp32-freertos.html#esp32-freertos)

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "sdkconfig.h"

void vTask1( void *pvParameters )
{
  const char *pcTaskName = "Task 1 is running";
  UBaseType_t uxPriority;
  uxPriority = uxTaskPriorityGet( NULL );
  for( ;; )
  {
    printf("%s - core = %d (priorite %d)\n", pcTaskName, xPortGetCoreID(), uxPriority);
    vTaskDelay( pdMS_TO_TICKS( 500 ) );
  }
}

void vTask2( void *pvParameters )
{
  const char *pcTaskName = "Task 2 is running";
  UBaseType_t uxPriority;
  uxPriority = uxTaskPriorityGet( NULL );
  for( ;; )
  {
    printf("%s - core = %d (priorite %d)\n", pcTaskName, xPortGetCoreID(), uxPriority);
    vTaskDelay( pdMS_TO_TICKS( 1000 ) );
  }
}

void app_main()
{
    xTaskCreate(vTask1, "vTask1", 10000, NULL, 1, NULL);
    xTaskCreate(vTask2, "vTask2", 10000, NULL, 1, NULL);

    for( ;; );
}
```

A

```
void vTask1( void *pvParameters )
{
  for( ;; )
  {
    Serial.printf("vTask1 %d\n", xPortGetCoreID());
    vTaskDelay( pdMS_TO_TICKS( 500 ) );
  }
}

void vTask2( void *pvParameters )
{
  for( ;; )
  {
    Serial.printf("vTask2 %d\n", xPortGetCoreID());
    vTaskDelay( pdMS_TO_TICKS( 500 ) );
  }
}

void setup()
{
  Serial.begin(115200);
  while (!Serial);
  Serial.println("Start");

  xTaskCreate(vTask1, "vTask1", 10000, NULL, 1, NULL);
  xTaskCreate(vTask2, "vTask2", 10000, NULL, 1, NULL);
}

void loop()
{
  Serial.printf("Task loop() %d\n", xPortGetCoreID());
  delay(1000);
}
```

FreeRTOS: Notations

Prefixe de type :

c : char

s : short

l : long

x : portBASE_TYPE

p : pointeur

v : void (fonction ne retournant aucun paramètre)

u : unsigned

uc : unsigned char

pc : pointeur sur un char ...

portCHAR : char (entier sur 8bits)

portSHORT : short (entier sur 16bits)

portLONG : long (entier sur 64bits)

portBASE_TYPE : int (entier sur 32bits)

Exemples :

xSemaphore : variable avec le type de base "portBASE_TYPE"

vTaskPrioritySet() retourne un ***void*** et est définie dans ***task.c***

xQueueReceive() retourne un ***portBASE_TYPE*** et est définie dans ***Queue.c***

vSemaphoreCreateBinary() retourne un ***void*** et est définie dans ***semphr.h***

FreeRTOS: création de tâche

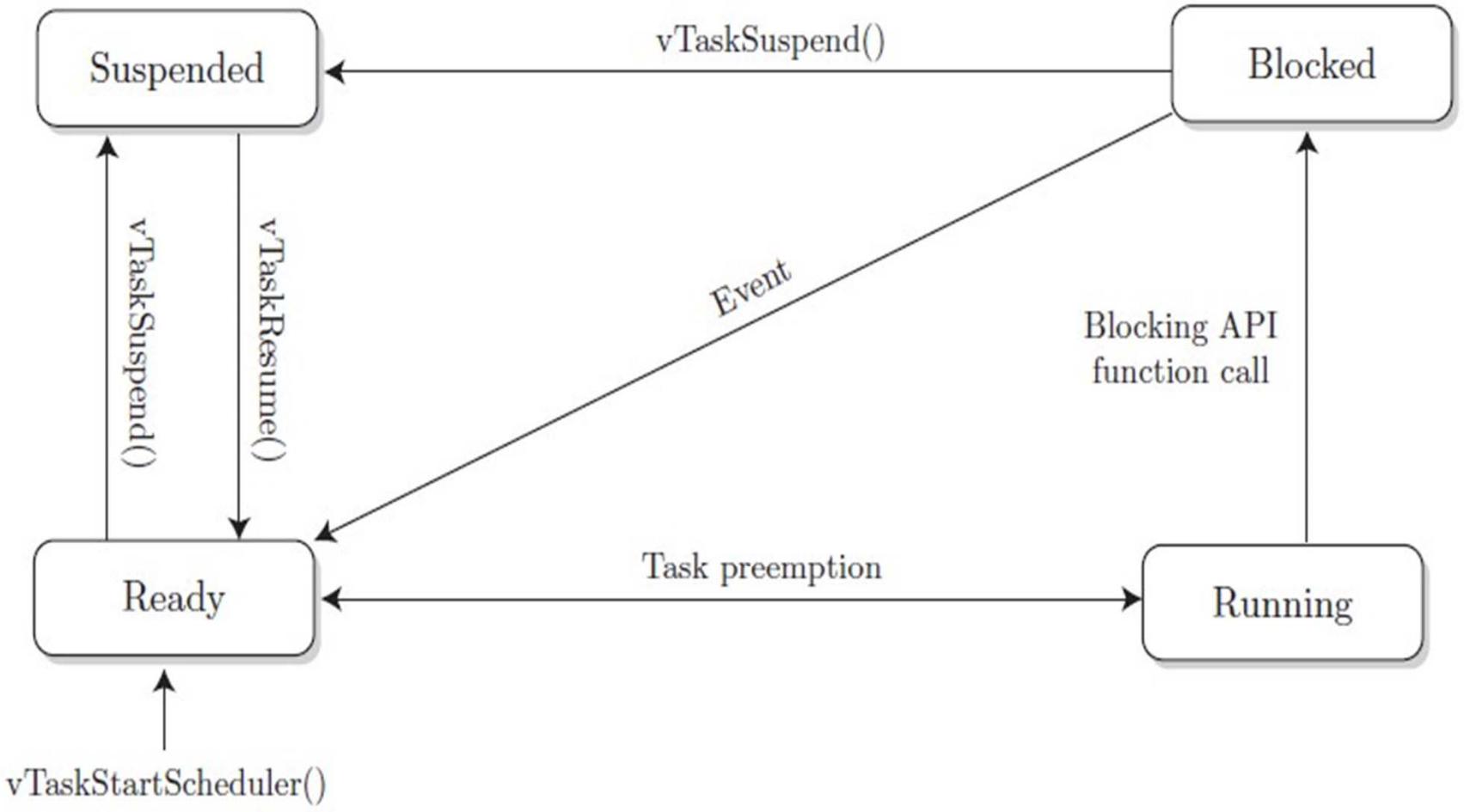
On crée une tâche avec un appel à la fonction *xTaskCreate()*

Cette fonction accepte les arguments suivants :

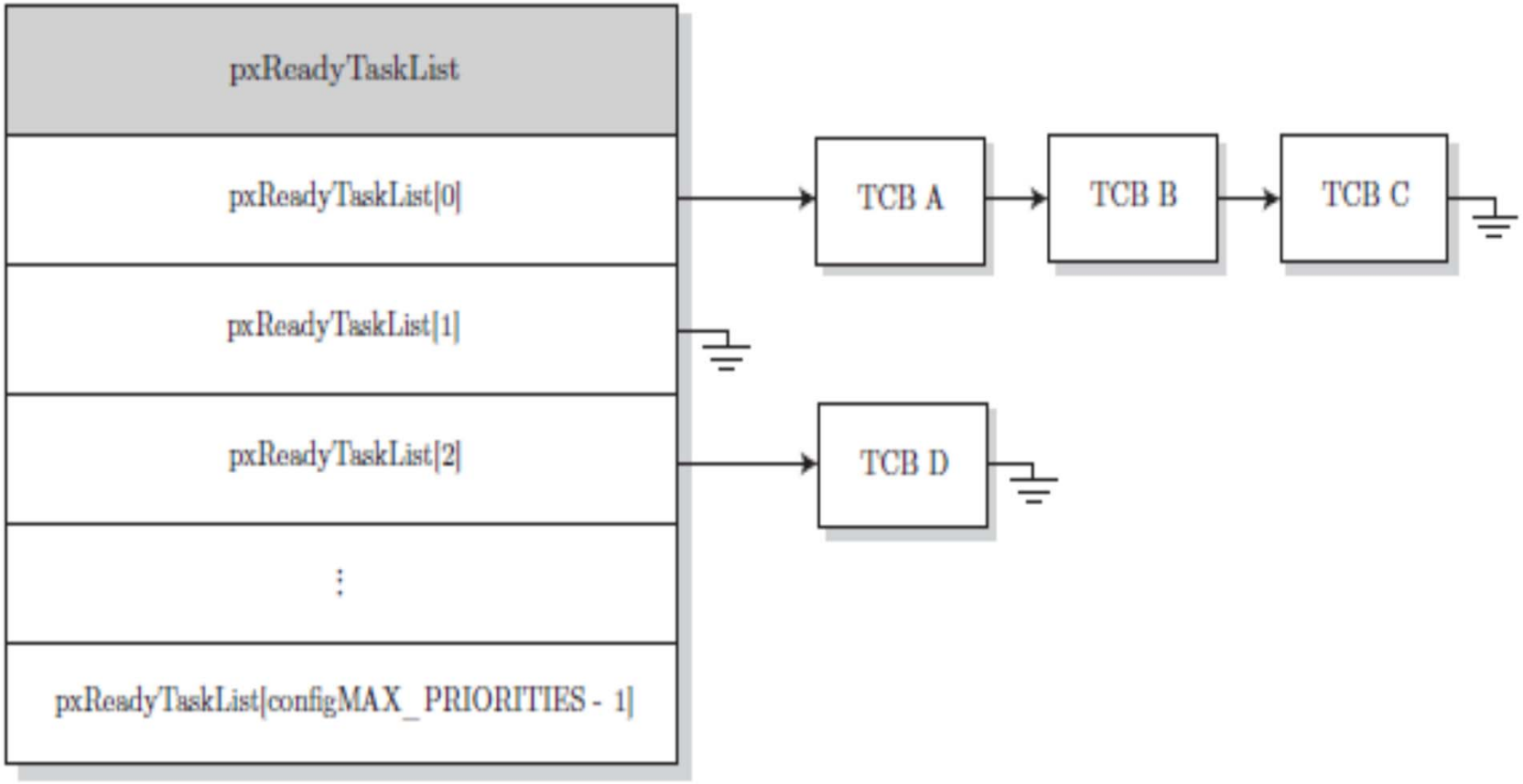
- *pvTaskCode* : pointeur sur la fonction qui implémentera la tâche (ici vATaskFunction)
- *pcName* : nom de la tâche (chaîne de caractères)
- *usStackDepth* : taille (en nombre de mots) de la pile
- *pvParameters* : pointeur vers un paramètre passée à la fonction de tâche (void *)
- *uxPriority* : priorité de la tâche (int)
- *pxCreatedTask* : descripteur de la tâche

Cette fonction retourne *pdPASS* en cas de succès ou un code d'erreur .

FreeRTOS: Différentes états d'une tâche



FreeRTOS: Priorités



FreeRTOS: Gestion Mémoire : Task Control Block (TCB)

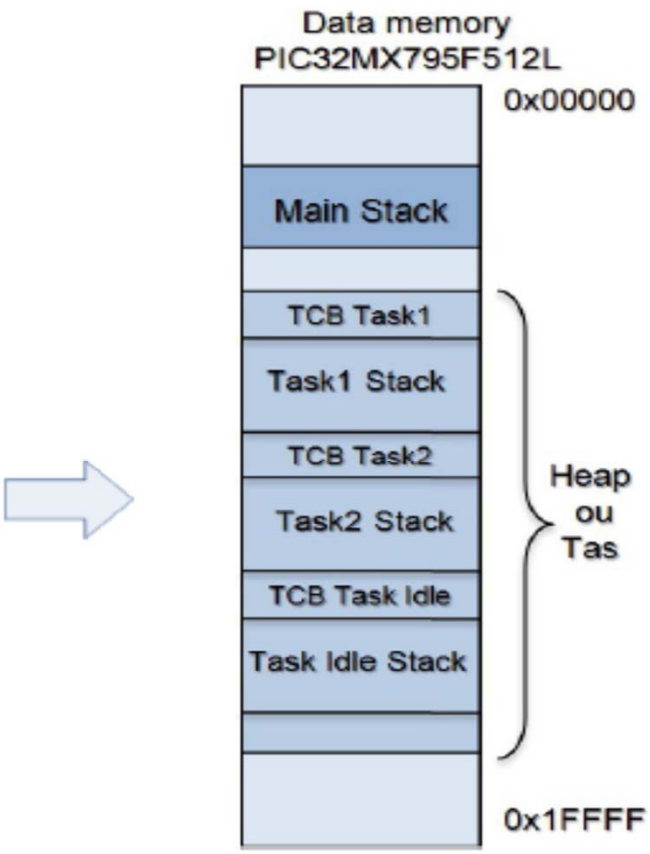
```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "sdkconfig.h"

void vTask1( void *pvParameters )
{
  const char *pcTaskName = "Task 1 is running";
  UBaseType_t uxPriority;
  uxPriority = uxTaskPriorityGet( NULL );
  for( ;; )
  {
    printf("%s - core = %d (priorite %d)\n", pcTaskName, xPortGetCoreID(), uxPriority);
    vTaskDelay( pdMS_TO_TICKS( 500 ) );
  }
}

void vTask2( void *pvParameters )
{
  const char *pcTaskName = "Task 2 is running";
  UBaseType_t uxPriority;
  uxPriority = uxTaskPriorityGet( NULL );
  for( ;; )
  {
    printf("%s - core = %d (priorite %d)\n", pcTaskName, xPortGetCoreID(), uxPriority);
    vTaskDelay( pdMS_TO_TICKS( 1000 ) );
  }
}

void app_main()
{
    xTaskCreate(vTask1, "vTask1", 10000, NULL, 1, NULL);
    xTaskCreate(vTask2, "vTask2", 10000, NULL, 1, NULL);

    for( ;; );
}
```



FreeRTOS: création de sémaphores

Les sémaphores FreeRTOS sont stockés dans une variable de type *SemaphoreHandle_t*

#include «freertos/semphr.h»

•Mutex: *xSemaphoreCreateMutex()* -> récursif + héritage de priorité!

•Sémaphore binaire: *xSemaphoreCreateBinary()*

•Sémaphore de comptage: *xSemaphoreCreateCounting(uxMaxCount, uxInitialCount)*

où *uxMaxCount* représente la valeur maximale de comptage et *uxInitialCount* sa valeur initiale.

« Prendre » *xSemaphoreTake()* ou *xSemaphoreTakeFromISR()*

un timeout peut être passé en second argument)

« Vendre » *xSemaphoreGive()* ou *xSemaphoreGiveFromISR()*

le niveau de priorité de relance de la tâche peut être passé en argument (ISR)

FreeRTOS: Input - Output

Sélection de la direction et affectation de la sortie

Framework Arduino:

pinMode(pin, mode)

avec mode = OUTPUT, INPUT, INPUT_PULLUP, …

digitalWrite(pin, value = HIGH ou LOW)

Val=digitalRead(pin)

Framework Idf: -> #include "driver/gpio.h"

gpio_set_direction(GPIO_OUTPUT_IO_0, MODE);

avec MODE = GPIO_MODE_OUTPUT, …

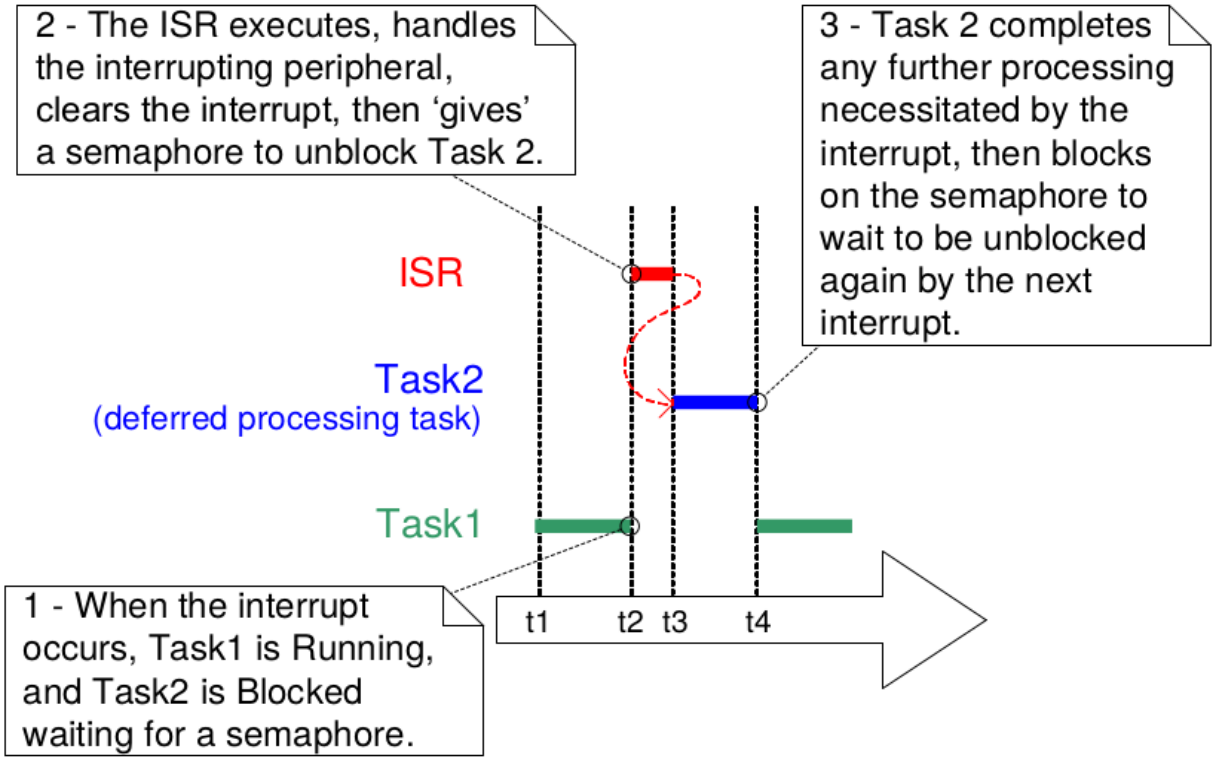
gpio_set_pull_mode(GPIO_INPUT_IO_0, MODEPULL)

avec MODEOULL = GPIO_PULLUP_ONLY, *…*

gpio_set_level(GPIO_OUTPUT_IO_0, HIGH ou LOW)

Val=gpio_get_level(GPIO_INPUT_IO_0)

FreeRTOS: Interruptions différées



Affectation du SSPI au vecteur d'interruption correspondant:

gpio_install_isr_service(0);

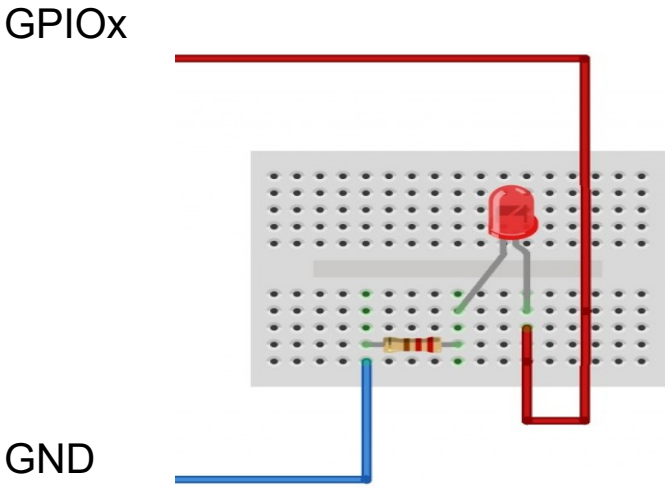
gpio_isr_handler_add(GPIO_INPUT_IO_0, gpio_isr_handler, (void*) GPIO_INPUT_IO_0);

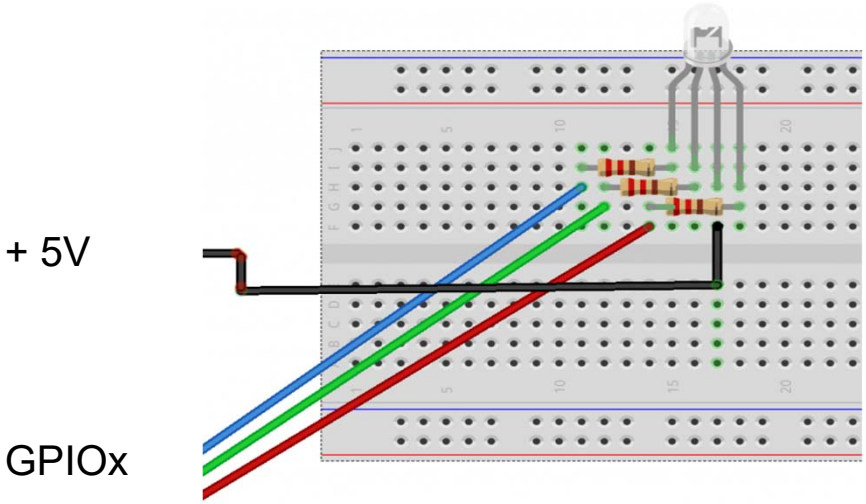
gpio_set_intr_type(GPIO_INPUT_IO_0, GPIO_INTR_ANYEDGE);

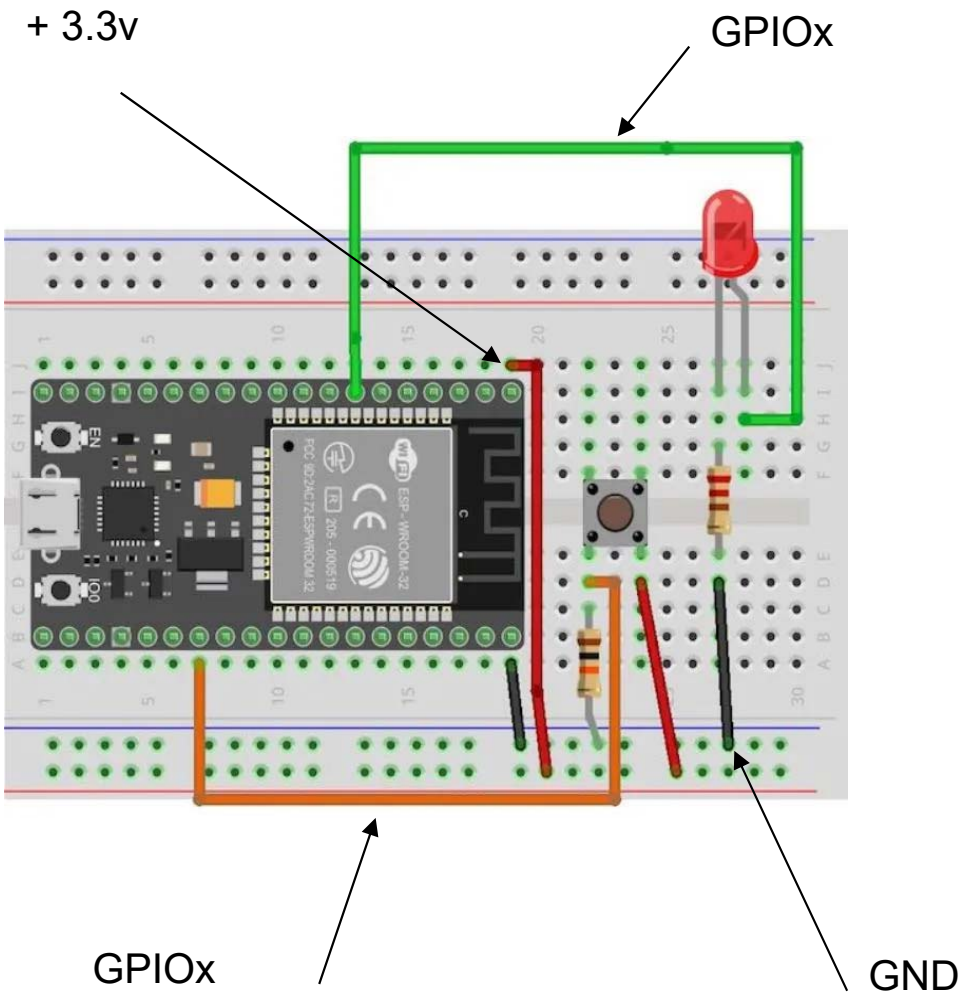
ou

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode); (Arduino)

Montages:







Sujet: Réalisation d'un modèle producteur unique – consommateurs multiples: