Université IBN TOFAIL – Kénitra
ECOLE NATIONALE DES SCIENCES APPLIQUEES

Cours Systèmes Temps Réel

CH5: Les Applications FreeRTOS

Par:

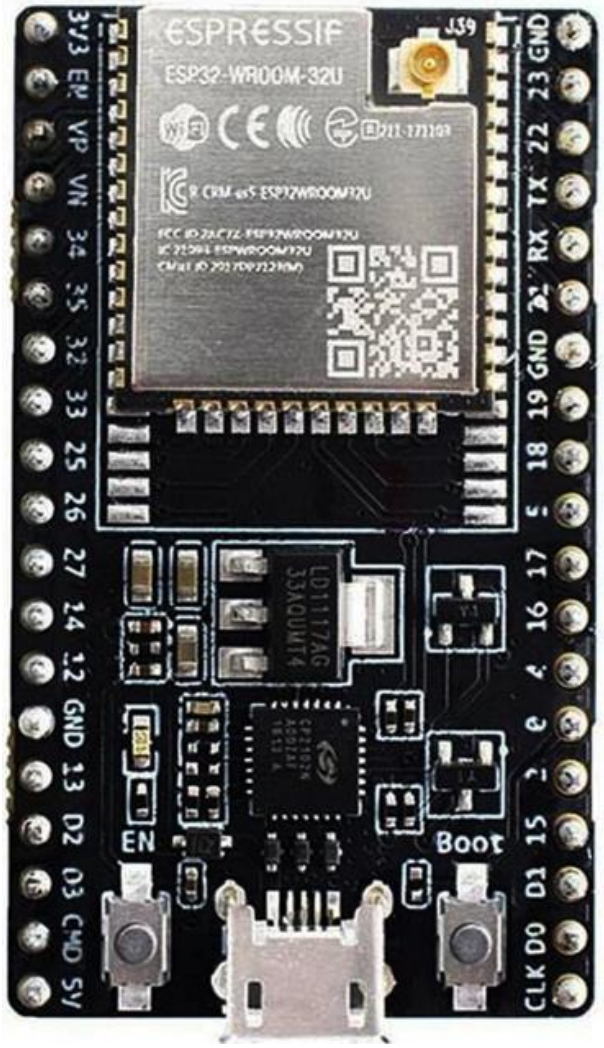
Hassan EL FADIL

elfadil.h@gmail.com

I- Utilisation de FreeRTOS avec la carte ESP32

1. Objectif

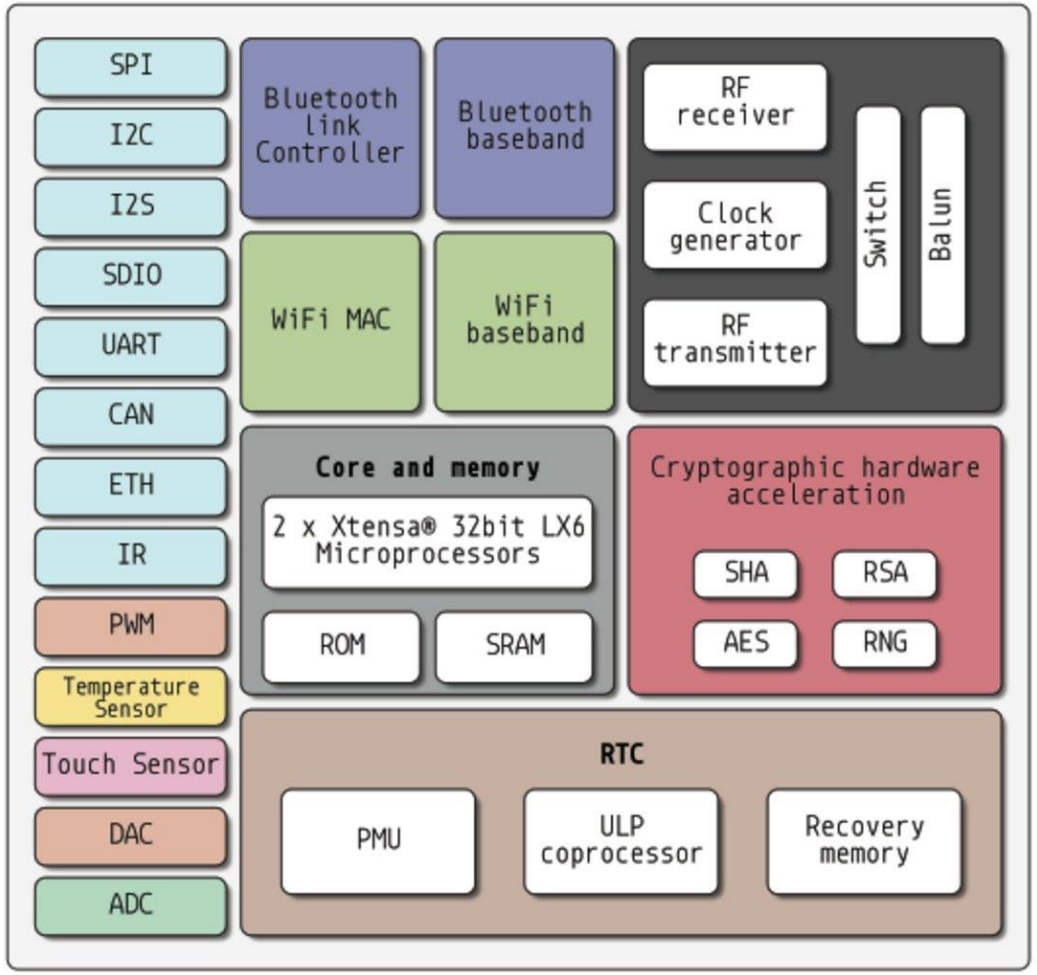
- Création de deux tâches avec des priorités différentes
- Utilisation de la carte **ESP32-WROOM-32**
- Utilisation de l’Arduino IDE



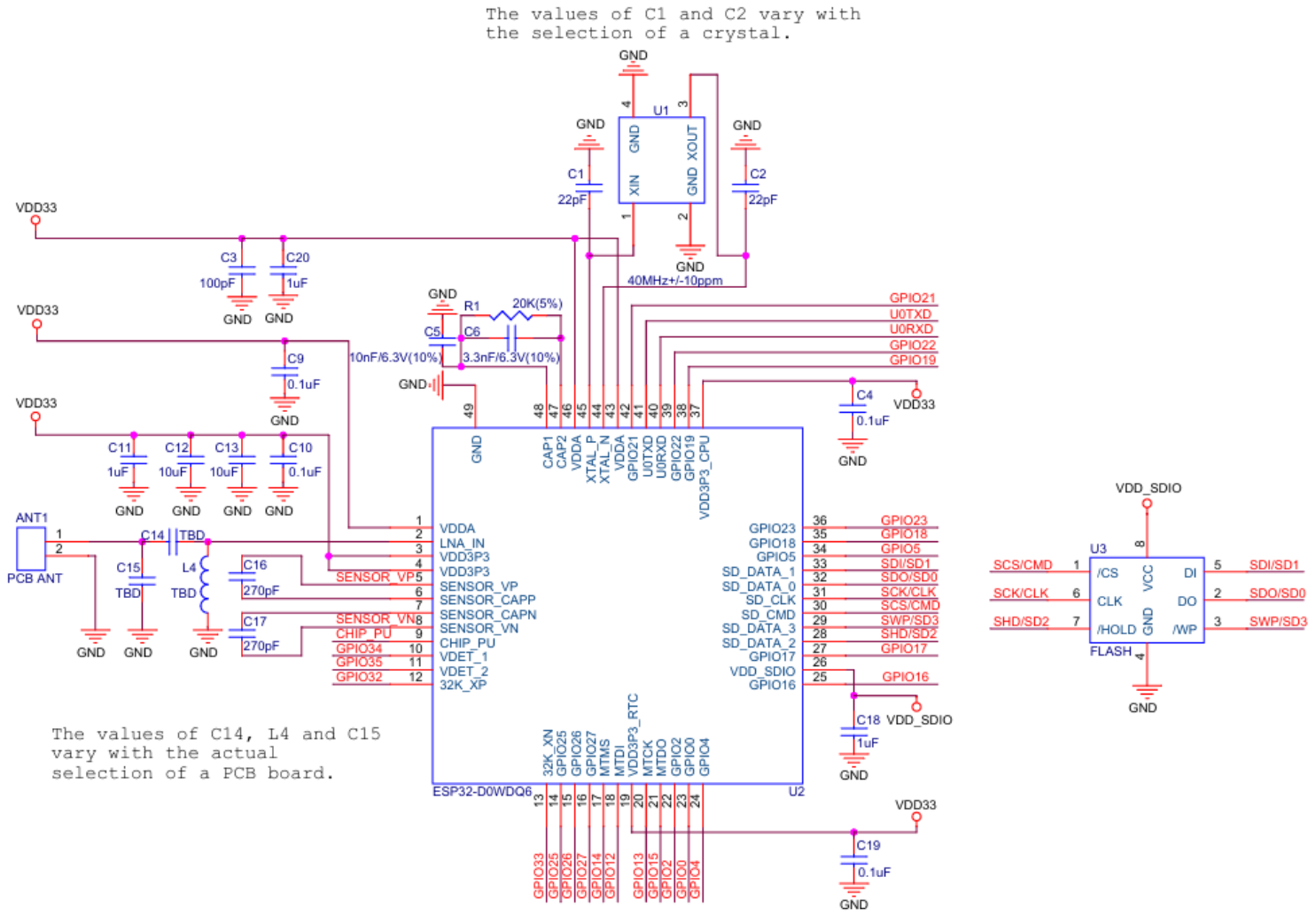
2. L'ESP-32-WROOM-32

2.1. Principales caractéristiques:

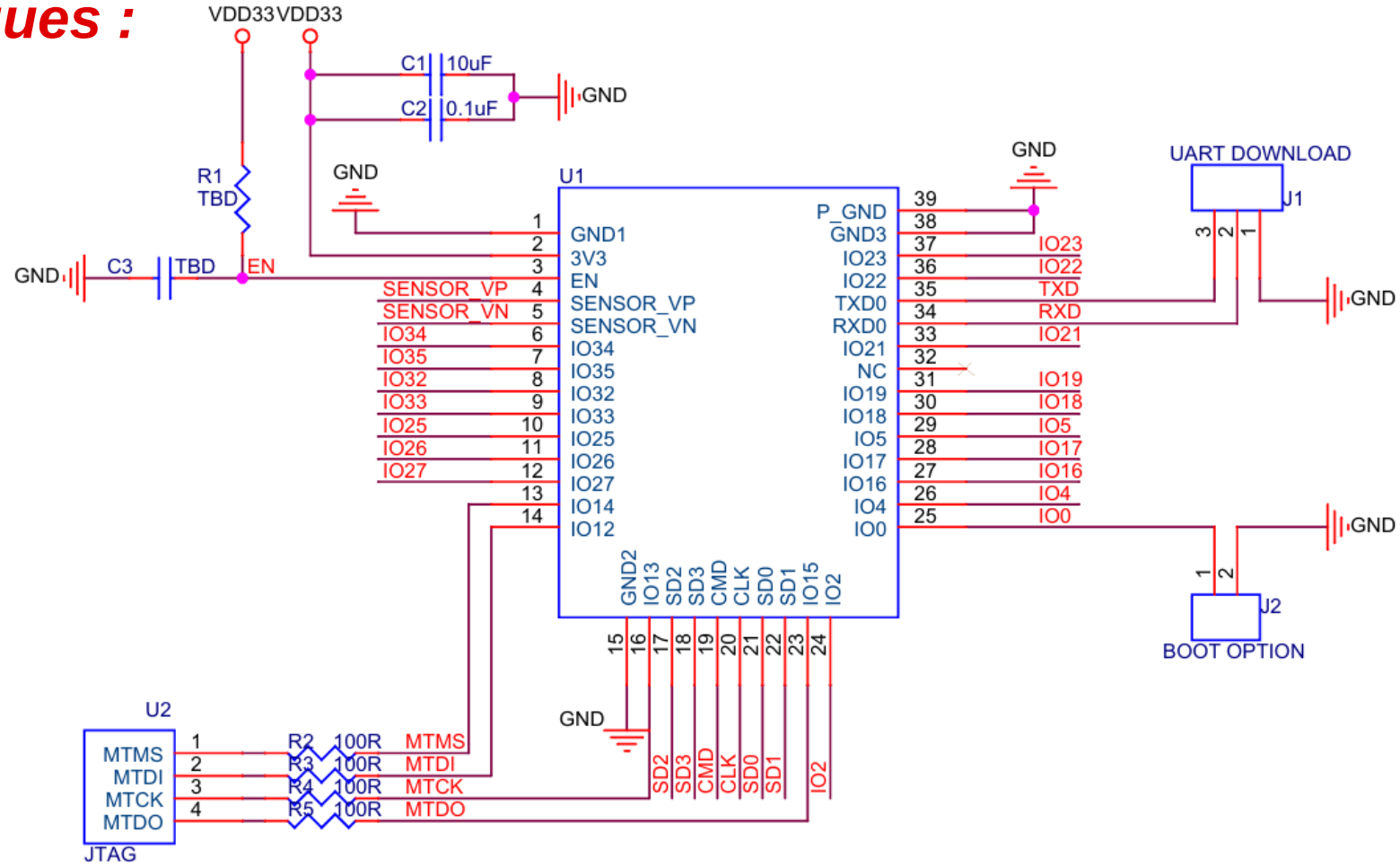
- **Microcontrôleur** : ESP32 dual-core Tensilica Xtensa LX6, cadencé jusqu'à 240 MHz.
- **WiFi** : IEEE 802.11 b/g/n, support pour les modes station, point d'accès et WiFi direct.
- **Bluetooth** : Bluetooth Low Energy (BLE) intégré, prenant en charge les profils BLE.
- **Mémoire** :
  - SRAM interne jusqu'à 520 KB.
  - Flash 4 Mo (SPI Flash). Jusqu'à 16Mo Selon la version
- **Interfaces** :
  - GPIO : Nombreuses broches GPIO pour les entrées/sorties.
  - UART, SPI, I2C : Interfaces série et bus de communication.
  - ADC: Jusqu'à 18 canaux ADC 12 bits.
  - DAC : 2 canaux DAC 8 bits.
  - PWM : Contrôle de la modulation de largeur d'impulsion pour la sortie analogique.



2.2. Schéma:


The values of C1 and C2 vary with the selection of a crystal.
The values of C14, L4 and C15 vary with the actual selection of a PCB board.
ESP32-D0WDQ6
U1
U2
U3
40MHz+/-10ppm
20K(5%)
3.3nF/6.3V(10%)
10nF/6.3V(10%)
PCB ANT
FLASH
VDD_SDIO
VDD33
GND

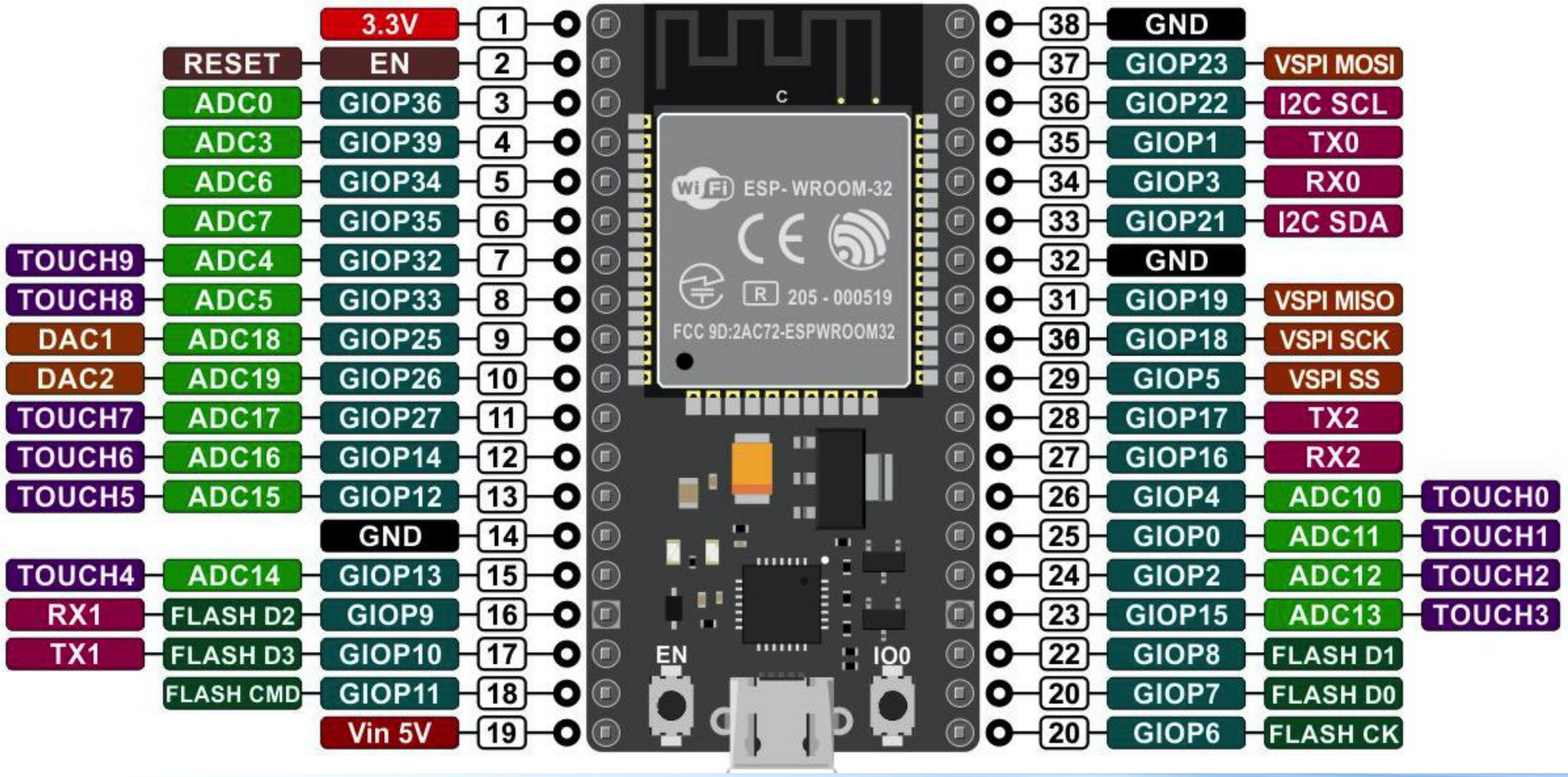
2.3. Périphériques :



MTDI should be kept at a low electric level when powering up the module.

2.4. Broches (Pins) :

Version
38 Pins

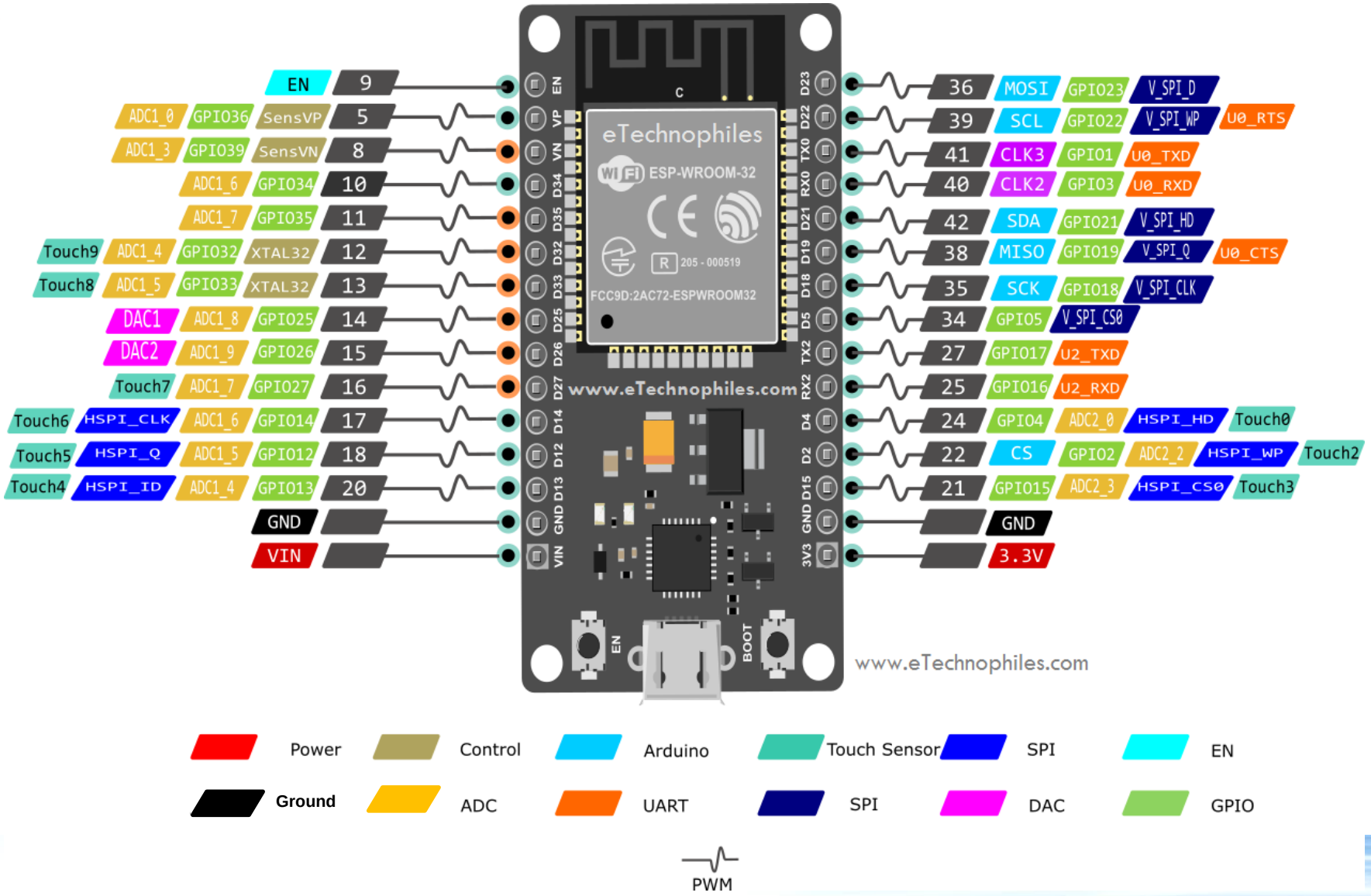


… ESP-32-WROOM-32 / Pinout

2.4. Broches (Pins):

Version 30 Pins:

Celle que nous allons utiliser



3. ESP32 et FreeRTOS

ESP32 WROOM est une architecture SMP :

Traitement symétrique multiprocesseur (SMP: Symmetric Multiprocessing):

- Microprocesseur double cœur Tensilica LX6. Chaque cœur fonctionne à une fréquence pouvant atteindre 240 MHz
- Les deux cœurs partagent le même espace mémoire
- L'ESP-IDF (**Espressif IoT Development Framework**) fournit un système d'exploitation en temps réel (FreeRTOS) qui prend en charge le SMP:

  https://docs.espressif.com/projects/esp-idf/en/release-v4.3/esp32/api-guides/freertos-smp.html

- Le planificateur FreeRTOS peut attribuer des tâches à l'un ou l'autre des cœurs, équilibrant la charge et améliorant les performances globales du système
- L'ESP32 inclut des mécanismes de communication et de synchronisation inter-processeurs, tels que les sémaphores et les files d'attente, essentiels pour coordonner les tâches entre les deux cœurs.
- L'ESP32 WROOM est conçu pour être économe en énergie, le rendant adapté aux applications alimentées par batterie

4. Configuration de l'Arduino IDE

Configurer le freeRTOS sur Arduino IDE

- Consulter des exemples sur GitHub:

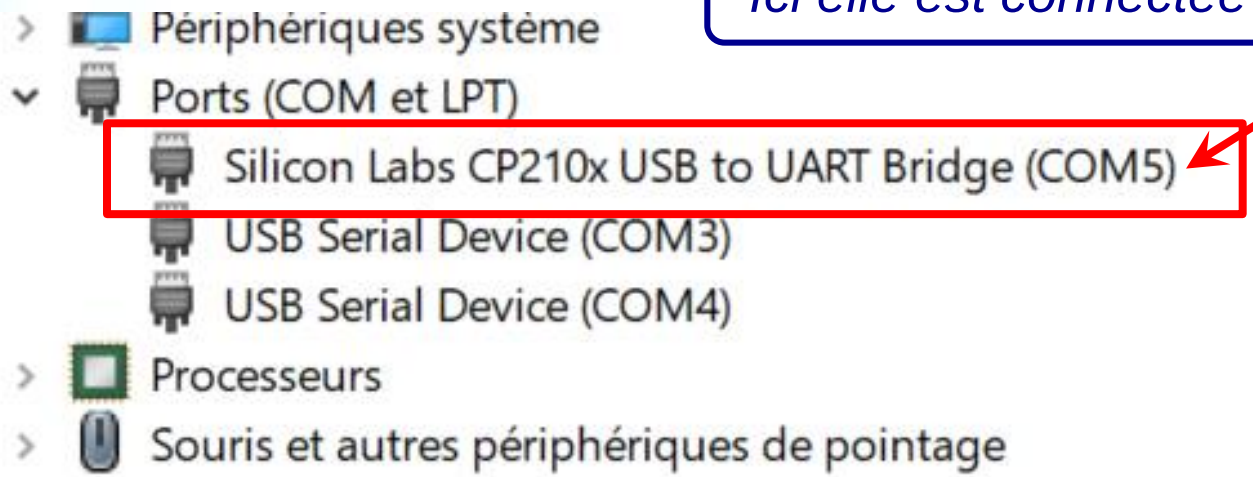
  https://github.com/ExploreEmbedded/Arduino_FreeRTOS

- A partir du gestionnaire de bibliothèque Arduino

1 **Connecter la carte ESP32 avec le PC (vérifier que le carte est bien détectée. Sinon installer ses drivers):**

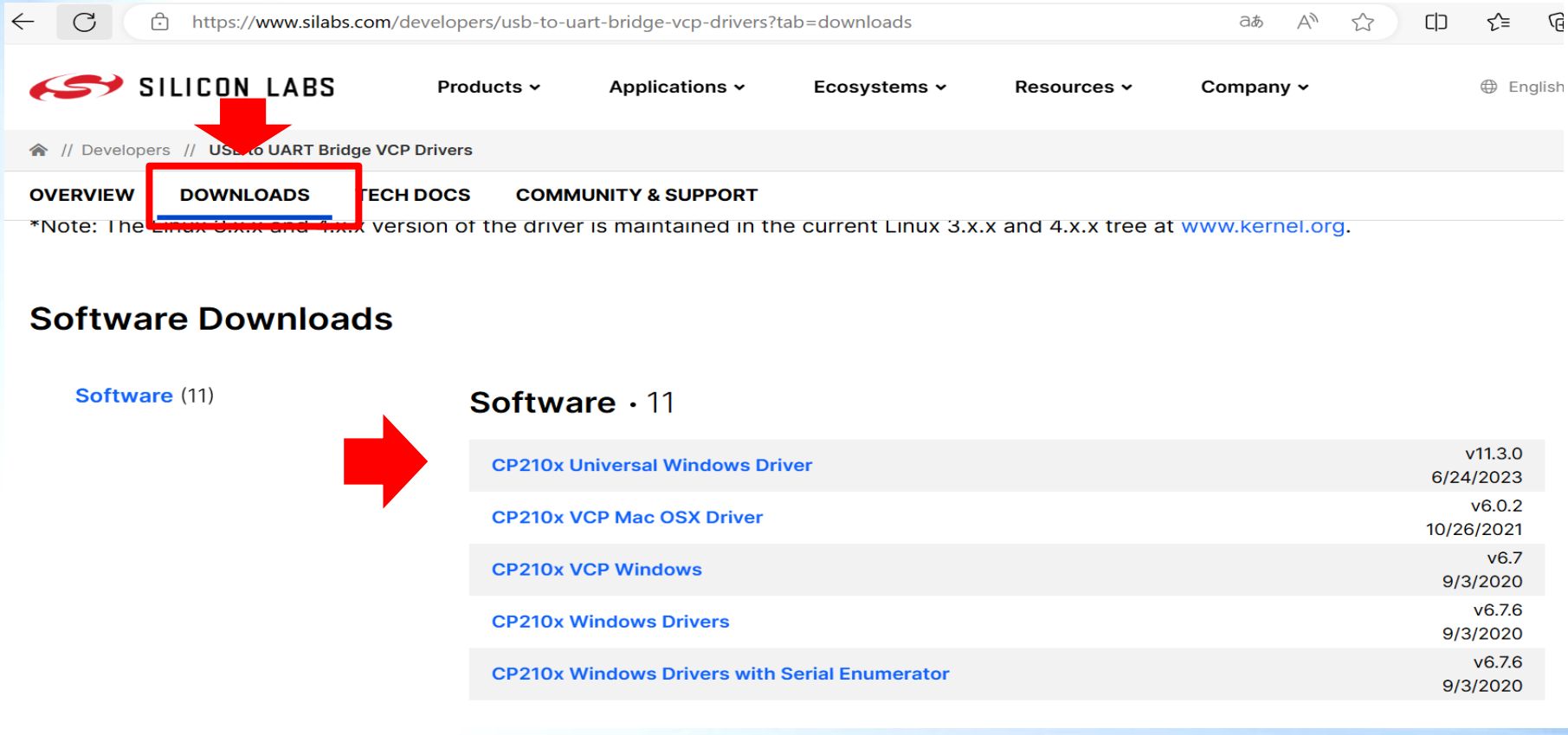
Ouvrir le gestionnaire de périphériques. Dans Ports (COM et LPT) vous devriez apercevoir la carte détectée.

*Ici elle est connectée sur le **COM5***



Pour installer le driver CP210x USB to UART Bridge:

https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers



Le **CP210x** USB to UART Bridge est un circuit intégré très utilisé pour convertir les signaux USB en signaux UART. Boîtiers carrés (comme les **QFP** - Quad Flat Package)

Le **CH340** pour boîtiers rectangulaires (**DIP**: Dual In-line Package)

2 Préparation de l'environnement de développement (Arduino IDE):

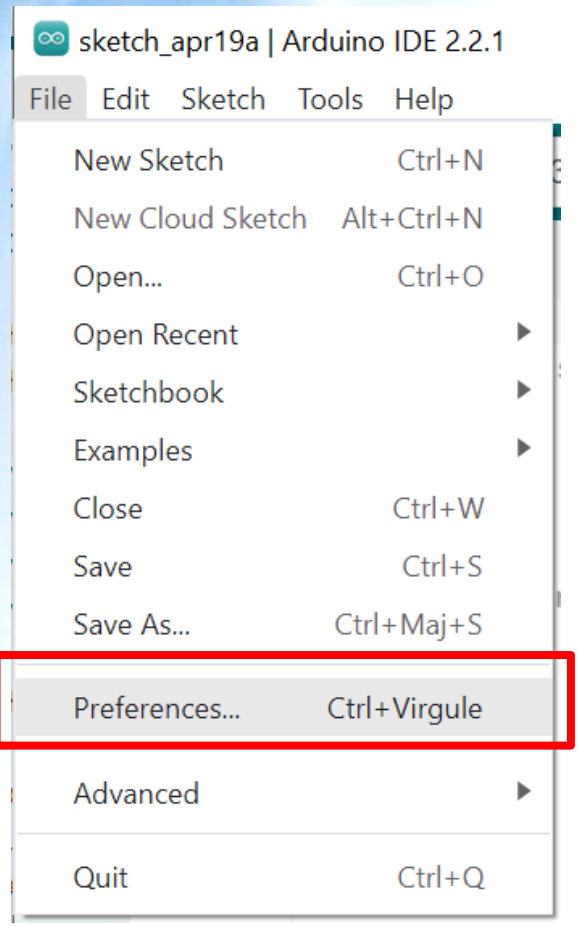
a) *Télécharger et Installer l'Arduino IDE sur votre ordinateur pour programmer l'ESP32*

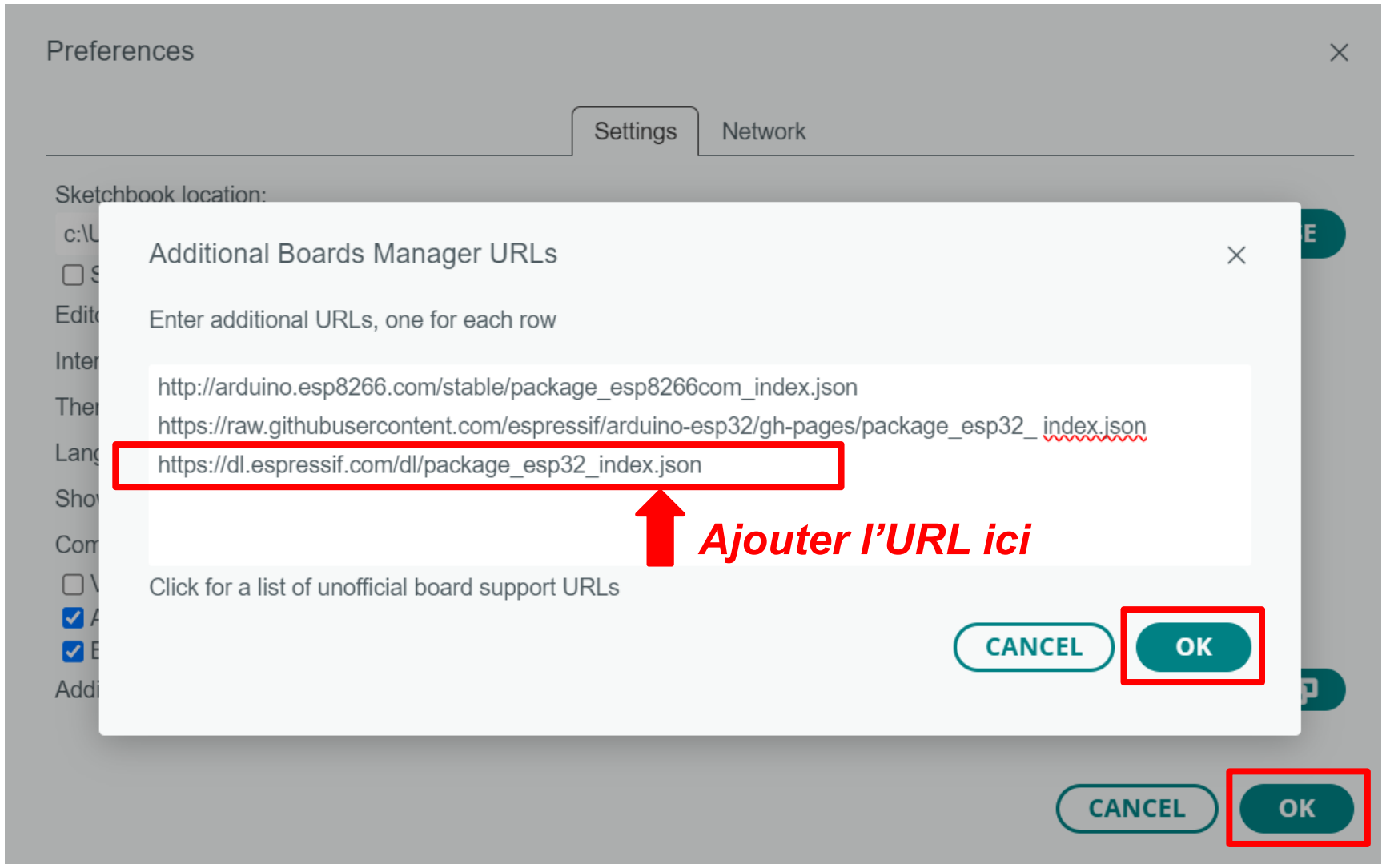
https://www.arduino.cc/.

b) *Ouvrir un nouveau fichier à partir de l'Arduino IDE.*
c) *Installer les packages de ESP32: Dans **préférences** ajouter l'URL suivant:*

https://dl.espressif.com/dl/package_esp32_index.json

… Configuration de l'Arduino IDE





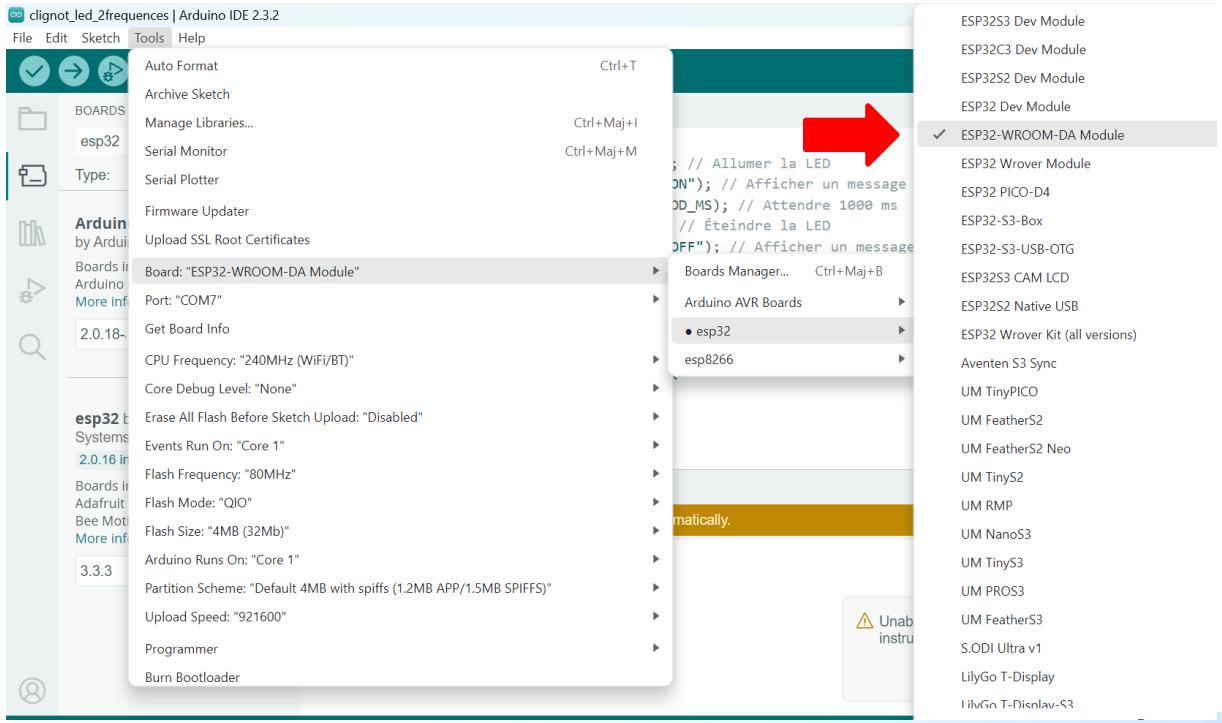
… Configuration de l'Arduino IDE

d) Dans "Tools", ouvrir "Boards Manager" :

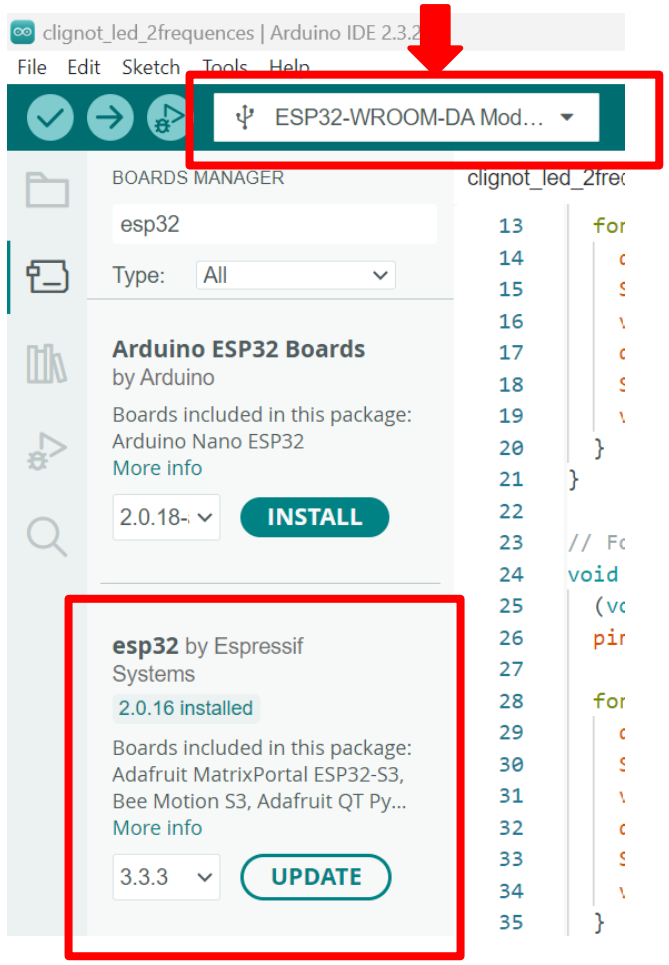
Dans le menu taper: **esp32**, puis installer ses packages

e) Une fois l'installation terminée, selectionner le WROOM-DA parmi les cartes ESP32:

Tools > Board > ESP32 > WROOM-DA Module



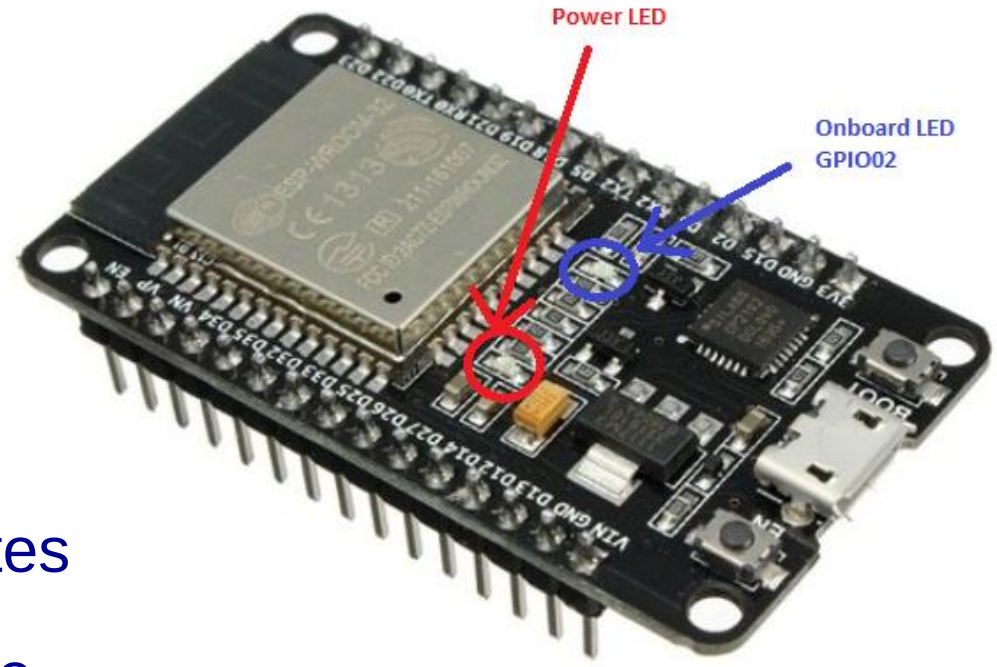
La carte de développement utilisée apparaît dans ce menu.



5. Code de l'application

Application:

- Clignotement de la **LED Bleue** de la carte ESP32-WROOM-32 (connectée au GPIO2)
- Utilisation de deux tâches différentes pour contrôler le clignotement
- Les tâches doivent avoir des priorités différentes
- La fréquence de clignotement de chaque tâche est différente



3 Code: Librairies et définition du GPIO:

```
#include <Arduino.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>

// Définir la broche de la LED. Utilisez la broche GPIO 2 pour la LED intégrée.
#define LED_BUILTIN 2
```

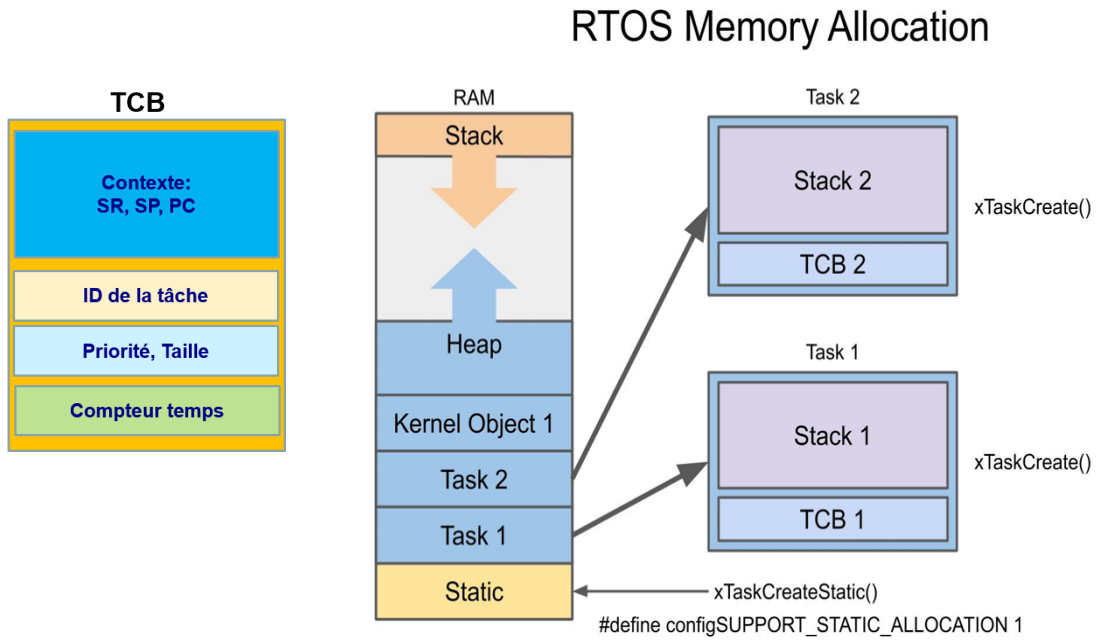
3 Code: Fonction de la 1ère tâche:

```
// Fonction de la tâche 1 : clignotement à 0.5 Hz
void TaskLED_1(void *pvParameters) {
  (void) pvParameters;
  pinMode(LED_BUILTIN, OUTPUT);

  for (;;) {
    digitalWrite(LED_BUILTIN, HIGH); // Allumer la LED
    Serial.println("TaskLED_1: LED ON"); // Afficher un message
    vTaskDelay(1000 / portTICK_PERIOD_MS); // Attendre 1000 ms
    digitalWrite(LED_BUILTIN, LOW); // Éteindre la LED
    Serial.println("TaskLED_1: LED OFF"); // Afficher un message
    vTaskDelay(1000/ portTICK_PERIOD_MS); // Attendre 1000 ms
  }
}
```

3 Code: Fonction de la 2ème tâche:

```
// Fonction de la tâche 2 : clignotement à 2.5 Hz
void TaskLED_2(void *pvParameters) {
  (void) pvParameters;
  pinMode(LED_BUILTIN, OUTPUT);

  for (;;) {
    digitalWrite(LED_BUILTIN, HIGH); // Allumer la LED
    Serial.println("TaskLED_2: LED ON"); // Afficher un message
    vTaskDelay(200 / portTICK_PERIOD_MS); // Attendre 200 ms
    digitalWrite(LED_BUILTIN, LOW); // Éteindre la LED
    Serial.println("TaskLED_2: LED OFF"); // Afficher un message
    vTaskDelay(200 / portTICK_PERIOD_MS); // Attendre 200 ms
  }
}
```

3 Code: Fonction setup:




```
void setup() {
  Serial.begin(9600); // Démarre la communication série à 9600 bauds
  // Créer les deux tâches avec des priorités différentes
  xTaskCreatePinnedToCore(
    TaskLED_1,     // Fonction de la tâche 1
    "TaskLED_1",   // Nom de la tâche 1 pour le débogage
    1024,            // Taille de la pile (en mots), min 768
    NULL,            // Paramètre passé à la tâche (ici, aucun).
    1,               // Priorité de la tâche 1
    NULL,            // Handle de la tâche (pas utilisé ici)
    0);              // Noyau où la tâche doit être exécutée (0 ou 1)
  xTaskCreatePinnedToCore(
    TaskLED_2,   // Fonction de la tâche 2
    "TaskLED_2", // Nom de la tâche 2 pour le débogage
    1024,            // Taille de la pile (en mots), min 768
    NULL,            // Paramètre passé à la tâche (ici, aucun)
    2,               // Priorité de la tâche 2 (plus élevée que la tâche 1)
    NULL,            // Handle de la tâche (pas utilisé ici). Un handle permet d'interagir avec la tâche en cours de son exécution
    1);              // Noyau où la tâche doit être exécutée (0 ou 1)
   // vous n'avez pas besoin de Démarrer le scheduler vTaskStartScheduler() après la définition des tâches,
  // La bibliothèque FreeRTOS pour Arduino gère cela automatiquement
}
```

3 Code: Fonction loop:

```
void loop() {
  // Ne rien mettre ici. Le scheduler de FreeRTOS gère tout.
}
```

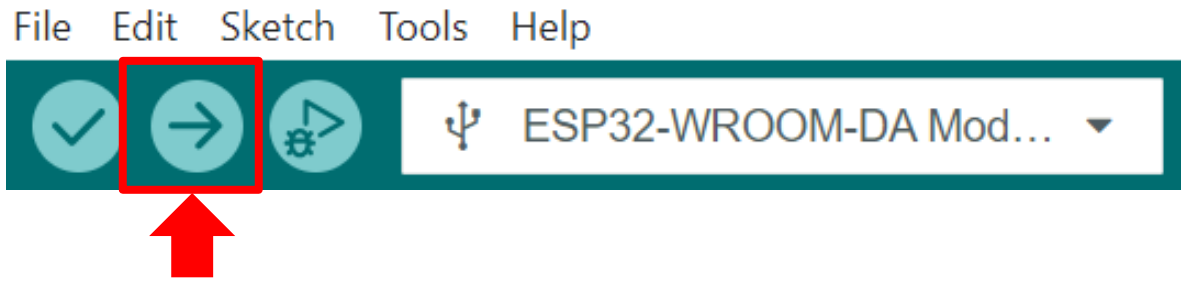
Explication la fonction: vTaskDelay

```
vTaskDelay(1000/ portTICK_PERIOD_MS);
```

- **vTaskDelay** : Cette fonction met la tâche en état de "bloquée" pendant une période de temps spécifiée. Pendant ce temps, la tâche ne s'exécutera pas et le système pourra exécuter d'autres tâches
- **1000**: Correspond à 1000 millisecondes, soit 1 seconde.
- **portTICK_PERIOD_MS**: Cette macro est définie dans FreeRTOS pour représenter la durée d'un tick en millisecondes. Par exemple, si le tick est de 1 ms, portTICK_PERIOD_MS vaut 1, si le tick est de 10 ms, portTICK_PERIOD_MS vaut 10.

6. Exécution et Tests

4 **La carte étant connectée au port USB du PC, Téléverser, alors, le programme (UPLOAD):**

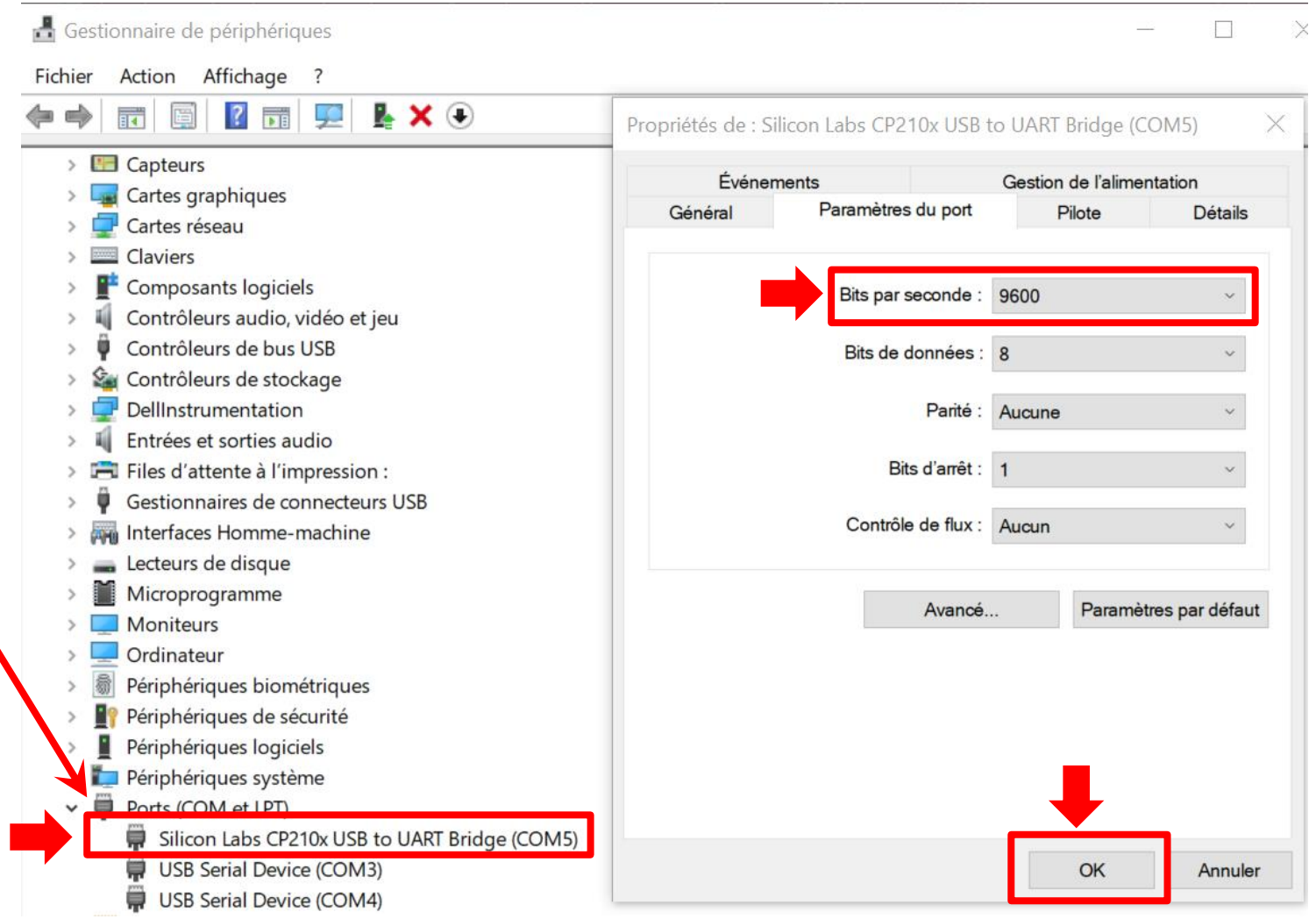


5 **Tester le programme:**

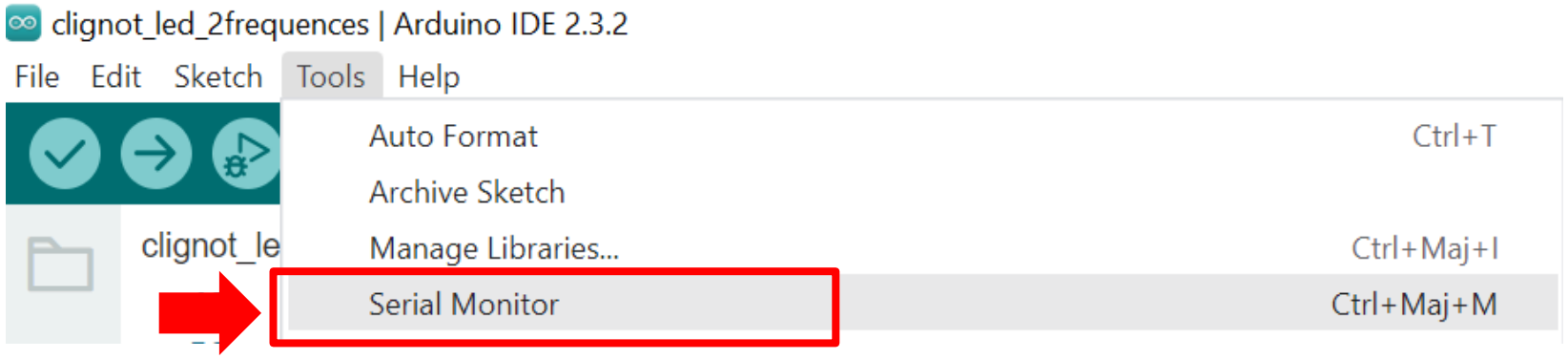
- Observer le clignotement de la LED intégrée à la carte.

6 Configurer le débit du port COM:

- Ouvrir le « Gestionnaire de périphériques »
- Ouvrir Ports (COM et LPT)
- Double-cliquer sur « Silicon Labs CP210x USB to UART Bridge »
- Régler le débit à **9600 bits/s**.

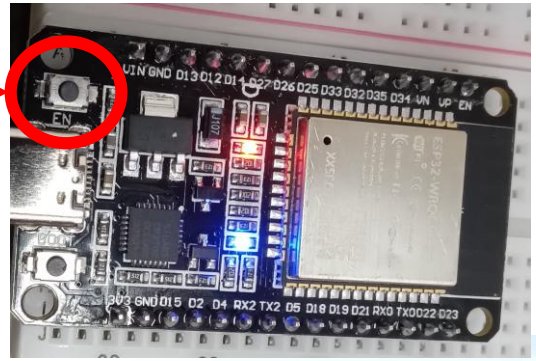


7 Ouvrir le Moniteur Série dans l'IDE Arduino: Tools > Serial Monitor



- Observer le défilement de l'exécution des tâches.
- Réinitialiser (reset) le microcontrôleur ESP32 et observer de nouveau l'exécution (**appui sur le bouton EN « Reset »**).

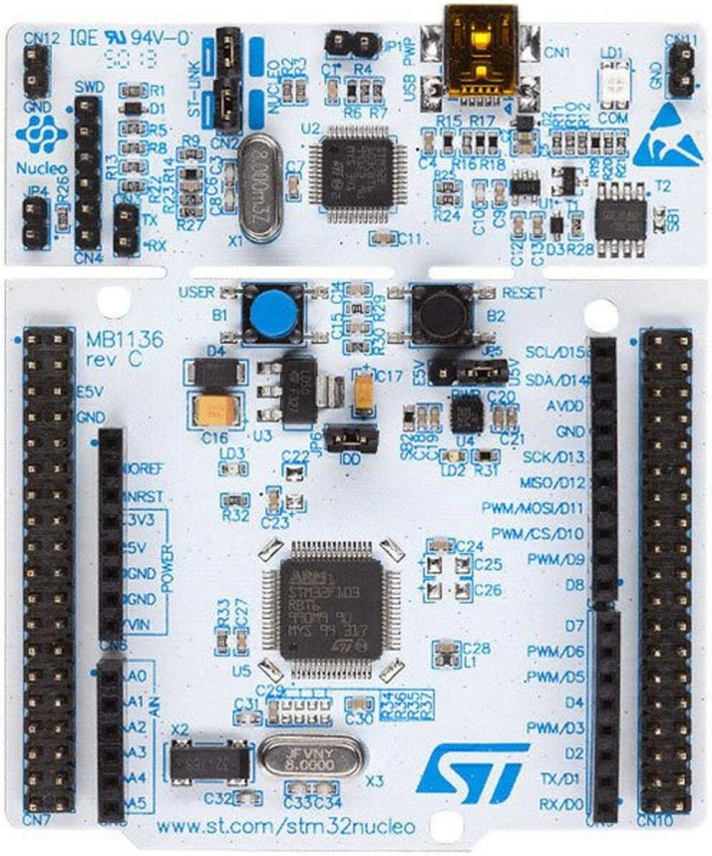
Bouton « Reset »



II- Utilisation de FreeRTOS avec la carte STM32

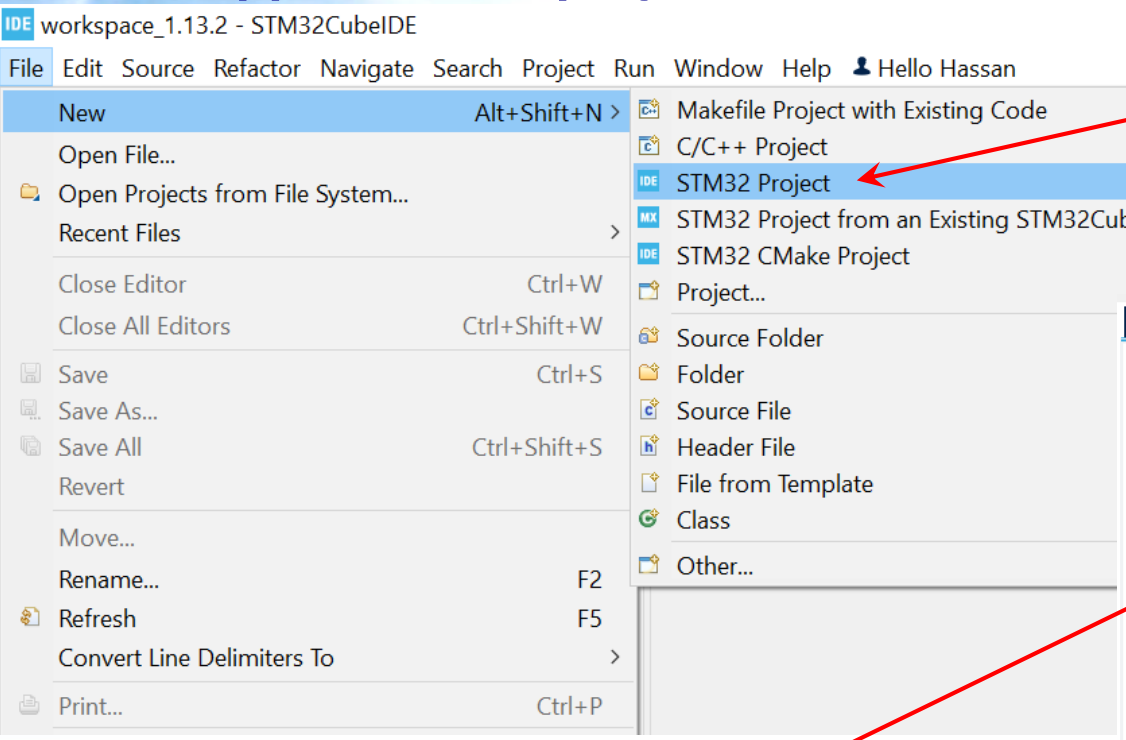
1. Objectif

- Création de deux tâches avec des priorités différentes
- Utilisation de la carte **NUCLEO-F401RE**
- Utilisation de l'IDE: **STM32CubeIDE**



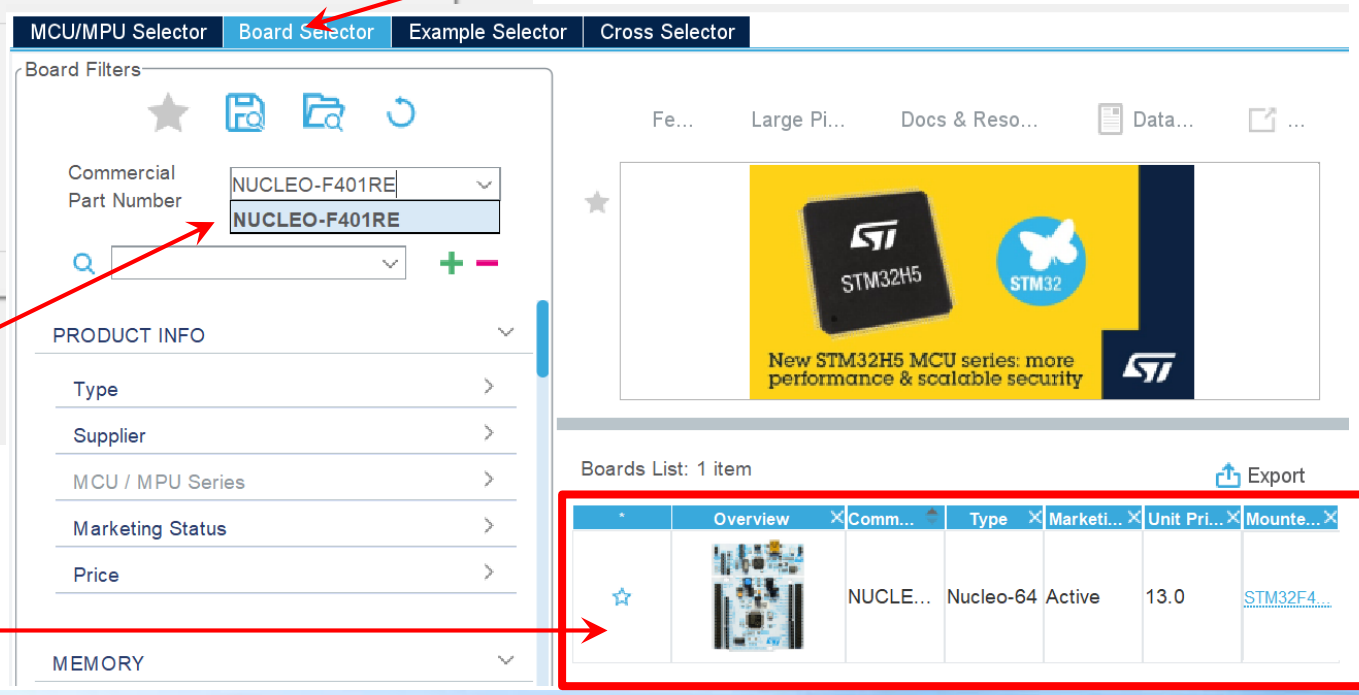
2. Création d'un projet STM32

1 Commencer par créer un nouveau projet STM32 (utiliser la carte NUCLEO-F401RE) Appeler votre projet: FreeRTOS_2T



Nouveau Projet STM32

Choisir « Board Selector »



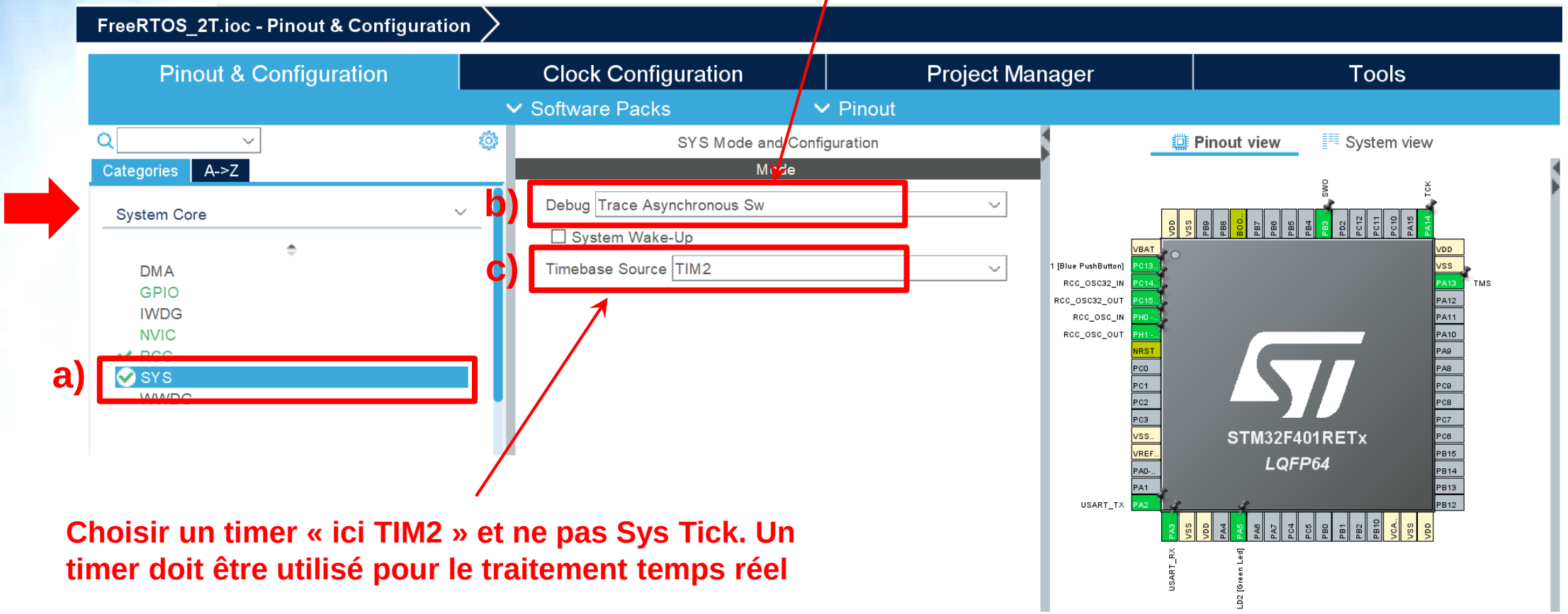
Chercher NUCLEO-F401RE

Sélectionner la carte et cliquer sur « Next »

3. Configuration de FreeRTOS

2 System Core > SYS > Mode

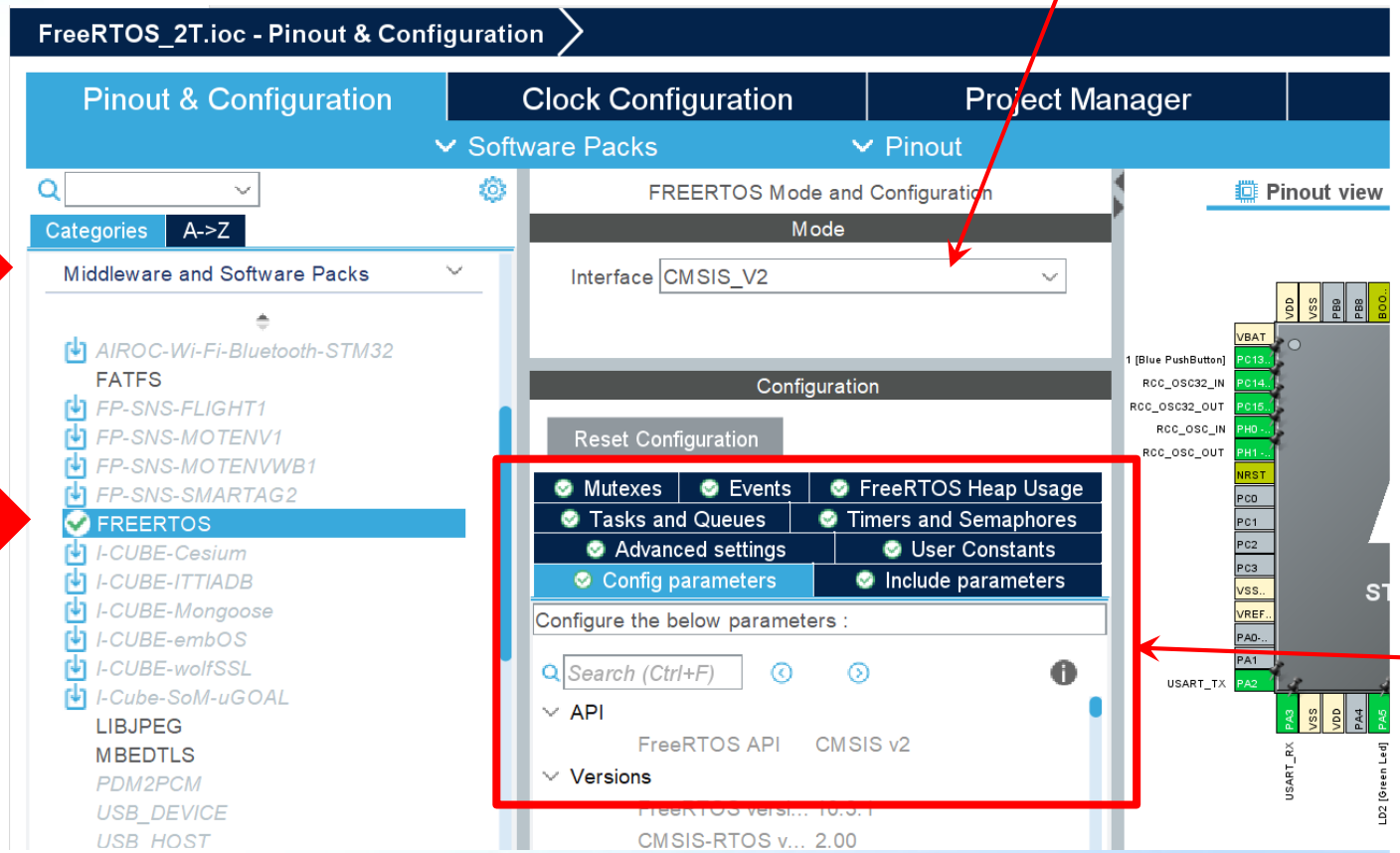
Choisir « Trace Asynchronous Sw » Cela est crucial pour le temps réel



Choisir un timer « ici TIM2 » et ne pas Sys Tick. Un timer doit être utilisé pour le traitement temps réel

3 Ensuite cliquer sur: Middleware and Sostware Packs > FREERTOS

Sélectionner « SMSIS_V2



a)

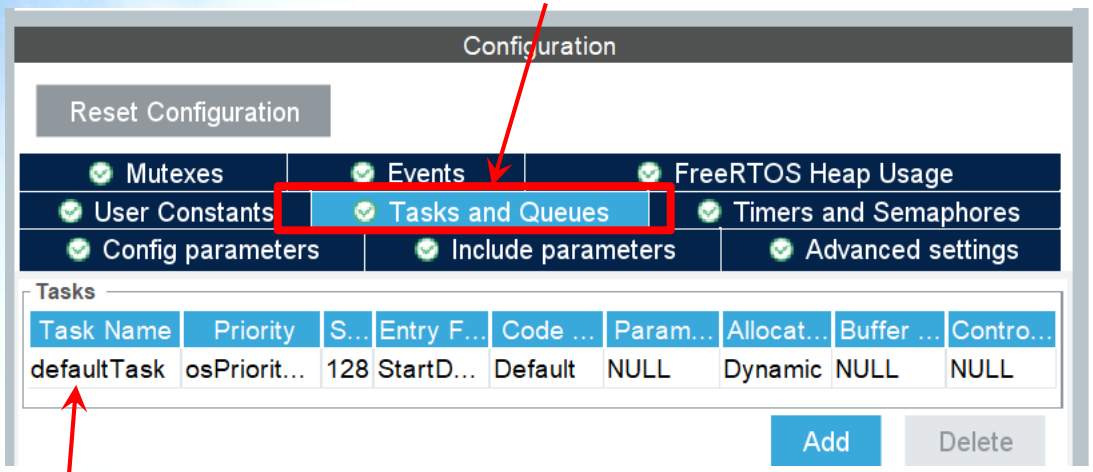
Middleware and Software Packs > FREERTOS

CMSIS-v2 (Cortex Microcontroller Software Interface Standard - API version 2): pour le développement d'applications embarquées, en particulier avec FreeRTOS et les microcontrôleurs STM32

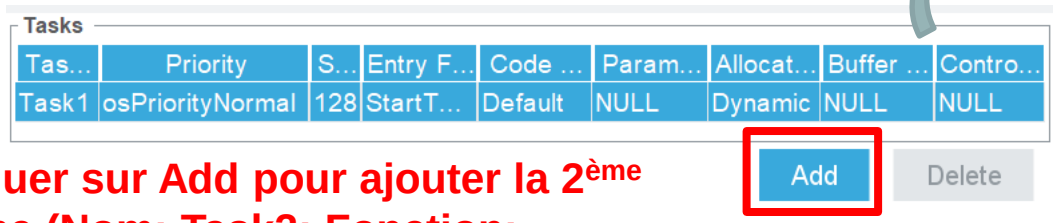
Vous pouvez constater ici les différents paramètres de configuration: les tâches et files d'attente, Heap, Mutexes, timers et sémaphores, …

4 Créer, ensuite, deux tâches: Task1 et Task2

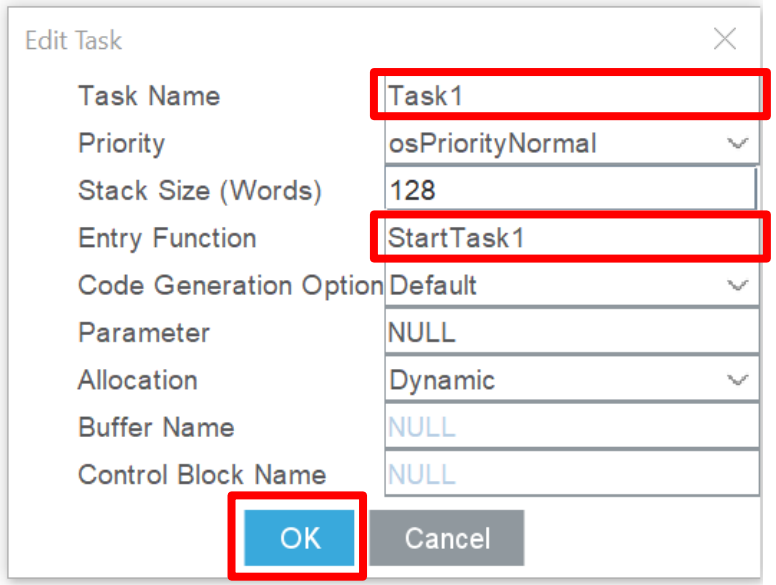
a) Sélectionner « Tasks and Queues »



b) Double-cliquer sur « defaultTask »



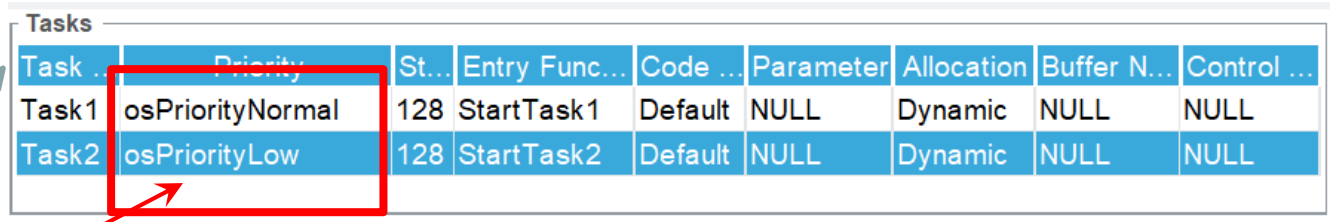
f) Cliquer sur Add pour ajouter la 2ème tâche (Nom: Task2; Fonction: StartTask2)



c) Nom: Task1

d) Fonction: StartTask1

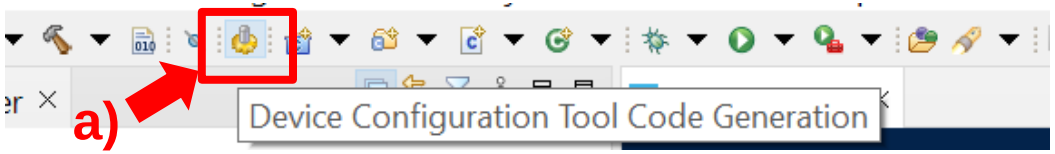
e) Cliquer sur OK



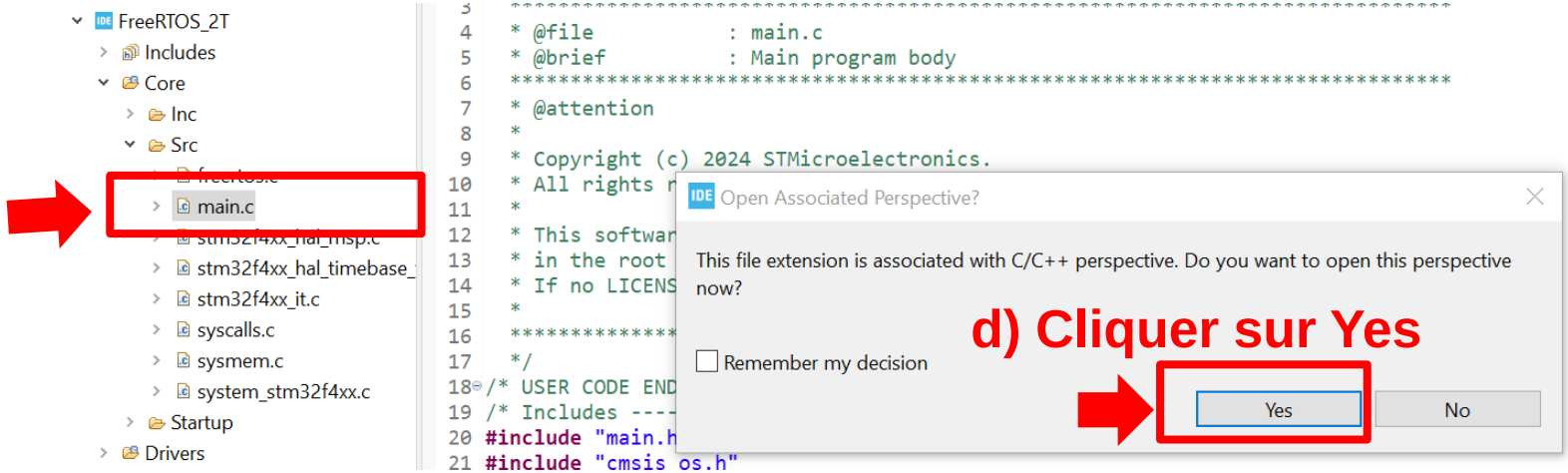
Remarquez que la tâche Task1 a un priorité normale et Task2 une priorité inférieure: c-à-d que T1 s'exécute en 1er et T2 après. Vous pouvez changer ces priorités selon l'ordre d'exécution désiré.

4. Génération du code

5 Cliquer, ensuite, « Device Configuration Tool Code Generation »



c) double-cliquer sur le fichier main.c



Le fichier main.c

a) Ici la définition des 2 tâches:

```
46 /* Definitions for Task1 */
47 osThreadId_t Task1Handle;
48 const osThreadAttr_t Task1_attributes = {
49   .name = "Task1",
50   .stack_size = 128 * 4,
51   .priority = (osPriority_t) osPriorityNormal,
52 };
53 /* Definitions for Task2 */
54 osThreadId_t Task2Handle;
55 const osThreadAttr_t Task2_attributes = {
56   .name = "Task2",
57   .stack_size = 128 * 4,
58   .priority = (osPriority_t) osPriorityLow,
59 };
```

b) Ici l'initialisation de l'ordonnanceur

```
113   /* Init scheduler */
114   osKernelInitialize();
115
```

c) Ici la création des 2 tâches:

```
133   /* creation of Task1 */
134   Task1Handle = osThreadNew(StartTask1, NULL, &Task1_attributes);
135
136   /* creation of Task2 */
137   Task2Handle = osThreadNew(StartTask2, NULL, &Task2_attributes);
```

d) Ici la fonction de démarrage de l'ordonnanceur

```
147   /* Start scheduler */
148   osKernelStart();
```

e) Constatez ici que la Super Loop (boucle while) est vide car il s'agit d'un RTOS

```
152   /* USER CODE BEGIN WHILE */
153   while (1)
154   {
155     /* USER CODE END WHILE */
156
157     /* USER CODE BEGIN 3 */
158   }
159   /* USER CODE END 3 */
```

f) Ici les fonctions des deux tâches: StartTask1 et StartTask2

```
289 void StartTask1(void *argument)
290 {
291   /* USER CODE BEGIN 5 */
292   /* Infinite loop */
293   for(;;)
294   {
295     osDelay(1);
296   }
297   /* USER CODE END 5 */
298 }
```

```
307 void StartTask2(void *argument)
308 {
309   /* USER CODE BEGIN StartTask2 */
310   /* Infinite loop */
311   for(;;)
312   {
313     osDelay(1);
314   }
315   /* USER CODE END StartTask2 */
316 }
```

C'est Ici où vous allez définir ce que les deux tâches doivent exécuter

Exemple:

lire les mesures de l'ADC, lire une entrée GPIO, générer un signal PWM, envoyer une sortie GPIO, etc!

Exemple d'étude:

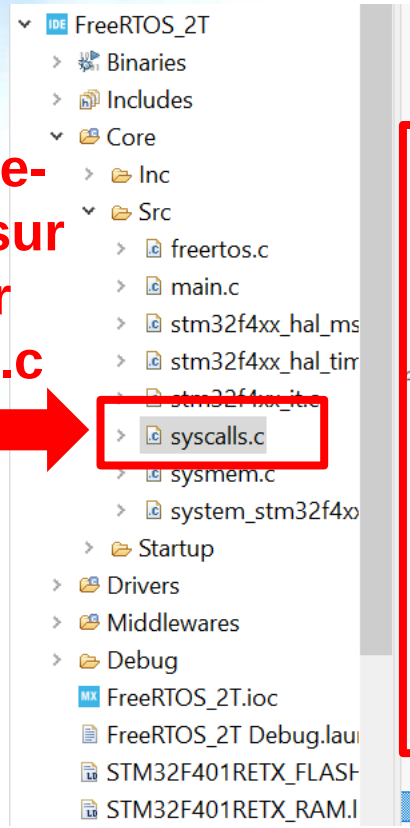
Dans cet exemple on souhaite afficher « Task-1 is running » quand la tâche 1 s'exécute et « Task-2 is running » quand la tâche 2 s'exécute.

Pour cela on aura besoin d'utiliser la fonction **printf**.

5. Configuration du fichier syscalls.c

6 Ajout d'un code au fichier syscalls.c pour supporter la fonction printf :

a) double-cliquer sur le fichier syscalls.c




```
28 #include <signal.h>
29 #include <time.h>
30 #include <sys/time.h>
31 #include <sys/times.h>
32
33 // Debug Exception & Monitor Control Register Base Address
34 #define DEMCR               (*((volatile uint32_t*) 0xE000EDFCU))
35 // ITM Register Addresses
36 #define ITM_TRACE_EN        (*((volatile uint32_t*) 0xE0000E00))
37 #define ITM_STIMULUS_PORT0  (*((volatile uint32_t*) 0xE0000000))
38
39 void ITM_SendChar(uint8_t ch) {
40     // Enable TRCENA in DEMCR (Debug Exception and Monitor Control Register)
41     DEMCR |= (1 << 24);
42
43     // Enable ITM stimulus port 0
44     ITM_TRACE_EN |= (1 << 0);
45
46     // Wait until the ITM stimulus port 0 is ready
47     while (!(ITM_STIMULUS_PORT0 & 1));
48
49     // Write the character to the ITM stimulus port 0
50     ITM_STIMULUS_PORT0 = ch;
51 }
52
53
54 /* Variables */
```

b) Ajouter ces lignes au code du fichier syscalls.c immédiatement après les #includes

… Configuration du fichier syscalls.c

Code à ajouter au fichier syscalls.c

```
// Debug Exception & Monitor Control Register Base Address
#define DEMCR (*((volatile uint32_t*) 0xE000EDFCU))
// ITM Register Addresses
#define ITM_TRACE_EN (*((volatile uint32_t*) 0xE0000E00))
#define ITM_STIMULUS_PORT0 (*((volatile uint32_t*) 0xE0000000))

void ITM_SendChar(uint8_t ch) {
// Enable TRCENA in DEMCR (Debug Exception and Monitor Control Register)
DEMCR |= (1 << 24);

// Enable ITM stimulus port 0
ITM_TRACE_EN |= (1 << 0);

// Wait until the ITM stimulus port 0 is ready
while (!(ITM_STIMULUS_PORT0 & 1));

// Write the character to the ITM stimulus port 0
ITM_STIMULUS_PORT0 = ch;
}
```

… Configuration du fichier syscalls.c

c) Modifier la fonction (weak write) dans le mêm fichier syscalls.c comme suit:

```
 99
100 __attribute__((weak)) int _write(int file, char *ptr, int len)
101 {
102   (void)file;
103   int DataIdx;
104
105   for (DataIdx = 0; DataIdx < len; DataIdx++)
106   {
107     //__io_putchar(*ptr++);
108     ITM_SendChar(*ptr++);
109   }
110   return len;
111 }
112
```

Commenter cette ligne (ajouter // au début de la ligne)

Ajouter cette ligne: `ITM_SendChar(*ptr++);`

d) Sauvegarder, enfin, le fichier (Ctrl+S)

6. Modification du code du fichier main.c

7 Ajouter printf aux fonctions des tâches Task1 et Task2 dans le fichier main.c:

a) Ajouter, d'abord, la bibliothèque standard d'entrée/sortie (stdio.h) dans la partie du code après USER CODE BEGIN Includes

```
23 /* Private includes ----------------
24 /* USER CODE BEGIN Includes */
25 #include <stdio.h>
26 /* USER CODE END Includes */
```

#include <stdio.h>

b) Modifier, ensuite, le code des Tâches (Task1 et Task2) en incluant printf :

```
288 /* USER CODE END Header_StartTask1 */
289 void StartTask1(void *argument)
290 {
291   /* USER CODE BEGIN 5 */
292   /* Infinite loop */
293   for(;;)
294   {
295     printf("Task-1 is running \n");
296     osDelay(1000); // Temporisation de 1s
297   }
298   /* USER CODE END 5 */
299 }
300
```

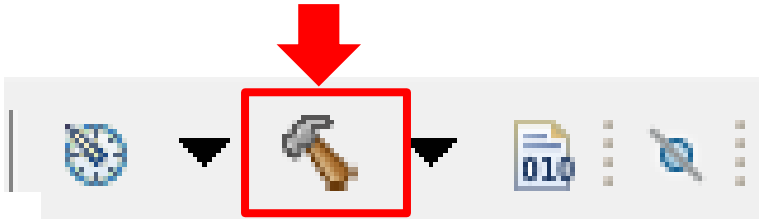
```
307 /* USER CODE END Header_StartTask2 */
308 void StartTask2(void *argument)
309 {
310   /* USER CODE BEGIN StartTask2 */
311   /* Infinite loop */
312   for(;;)
313   {
314     printf("Task-2 is running \n");
315     osDelay(1000); // Temporisation de 1s
316   }
317   /* USER CODE END StartTask2 */
318 }
319
```

```
printf("Task-1 is running \n");
osDelay(1000); // Temporisation de 1s

printf("Task-2 is running \n");
osDelay(1000); // Temporisation de 1s
```

c) Sauvegarder (Ctrl+S)

7. Compilation

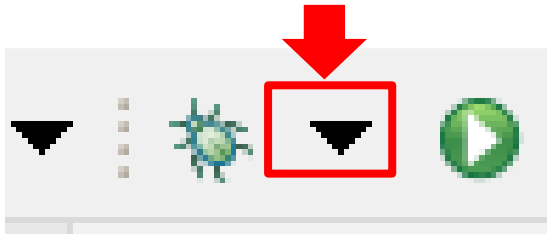
8 Lancer le Build pour vérifier les erreurs:



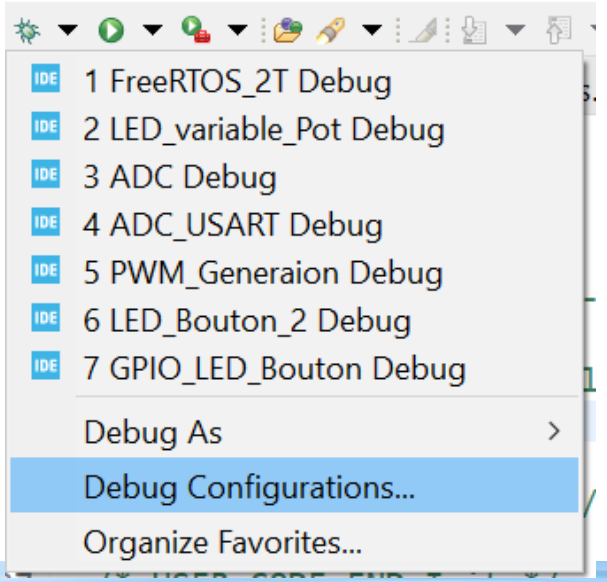
```
Finished building: default.size.stdout

19:30:33 Build Finished. 0 errors, 0 warnings. (took 278ms)
```

Vous devriez avoir un message indiquant 0 Erreurs

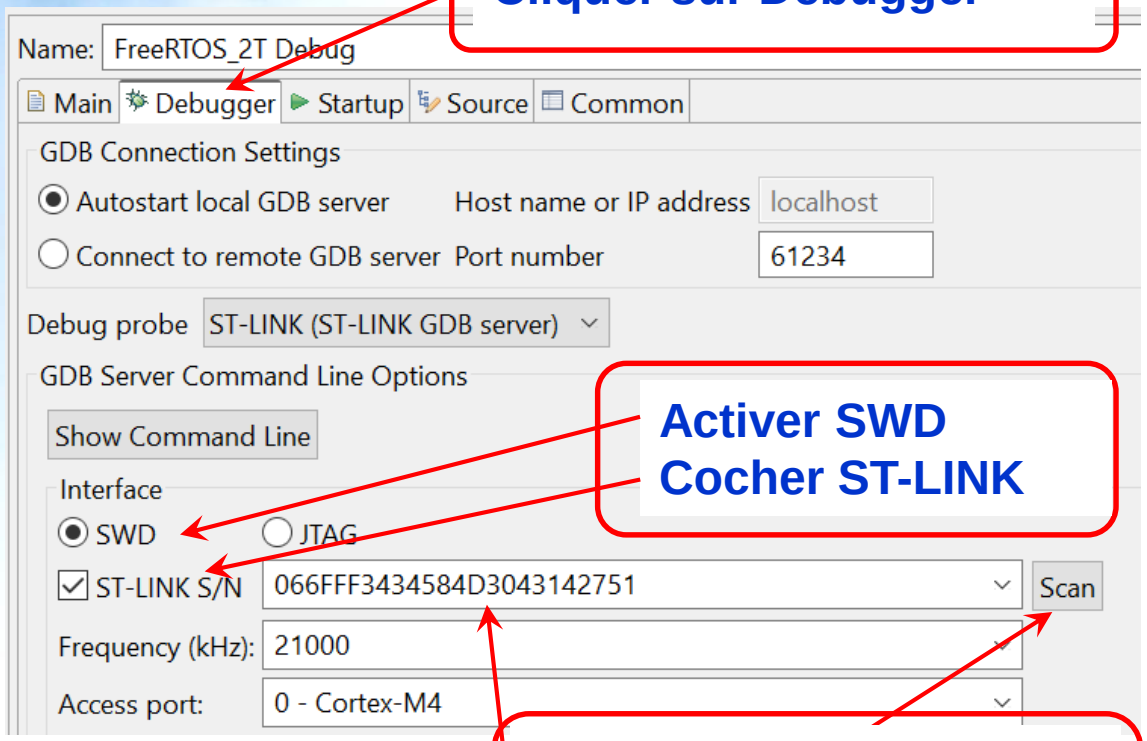
9 Configurer, ensuite, le Debug:

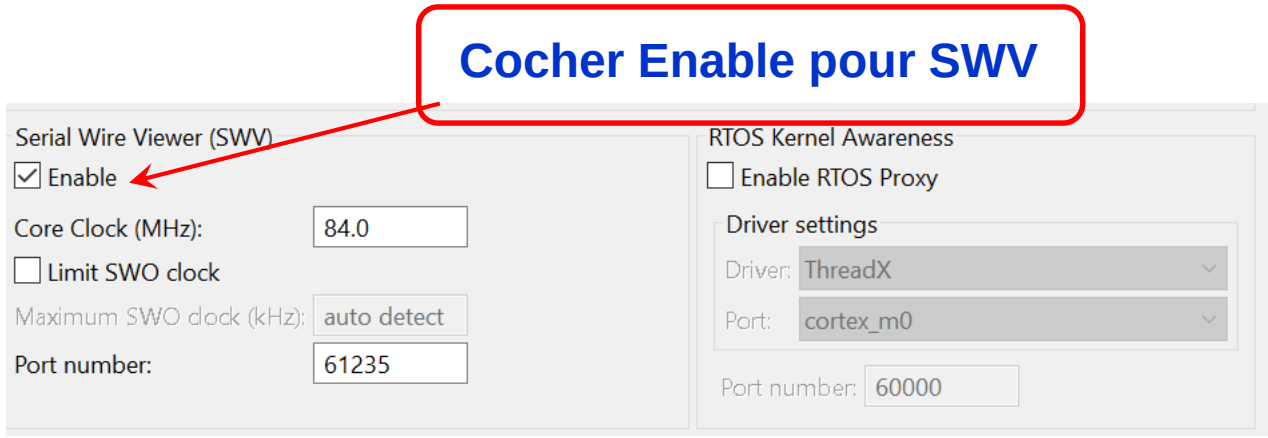


a) Selectionner « Debug Configuration »



b) Selectionner « Debug Configuration »

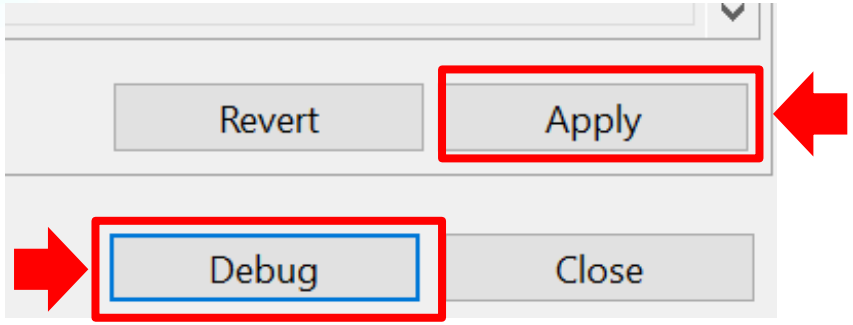


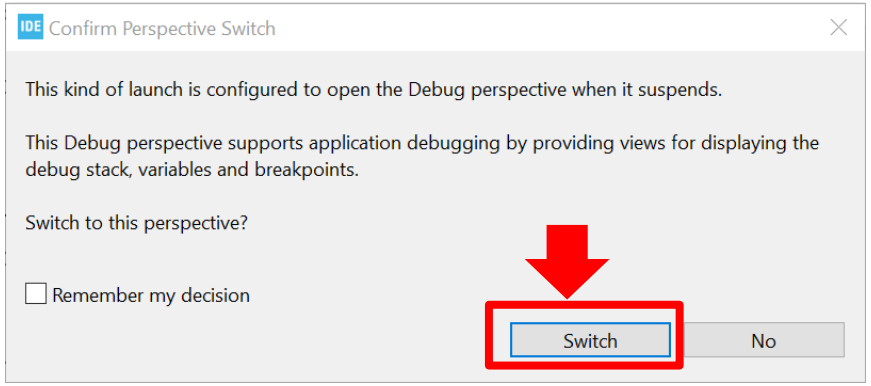


Le Serial Wire Viewer (SWV) est une fonctionnalité de débogage offerte par certains microcontrôleurs ARM Cortex-M qui permet de visualiser les données de trace en temps réel.

S'assurer que votre carte est connectée au port USB du PC (utiliser un câble mini USB type V3).

b) Terminer « Debug Configuration » par cliquer sur « Apply » puis « Debug »





Quand l'étape de Débogage est terminée vous apercevez:

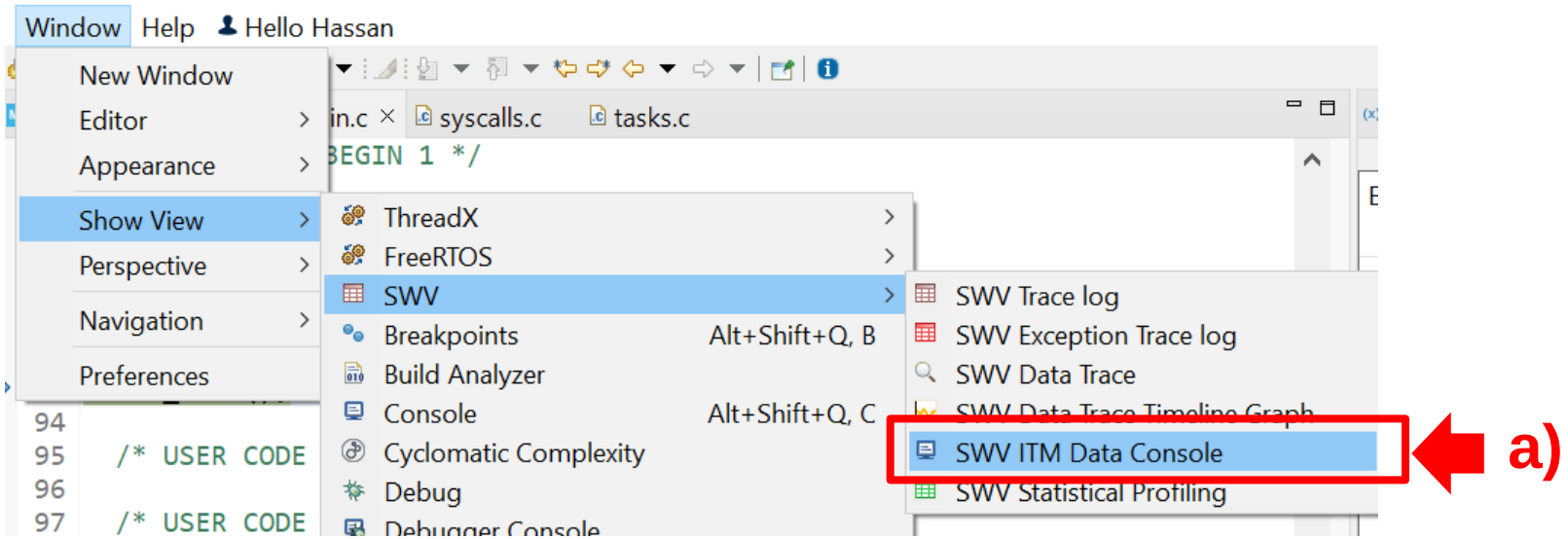
- **Un message « Download verified successfully**
- **Le programme s'arrêtte à la HAL_Init en surbrillance verte)**

```
90  /* MCU Configuration--------------------------------------------------------
91
92  /* Reset of all peripherals, Initializes the Flash interface and tl
93  HAL_Init();
94
95  /* USER CODE BEGIN Init */
96
97  /* USER CODE END Init */
```

Le programme est maintenant à l'étape HAL_Init

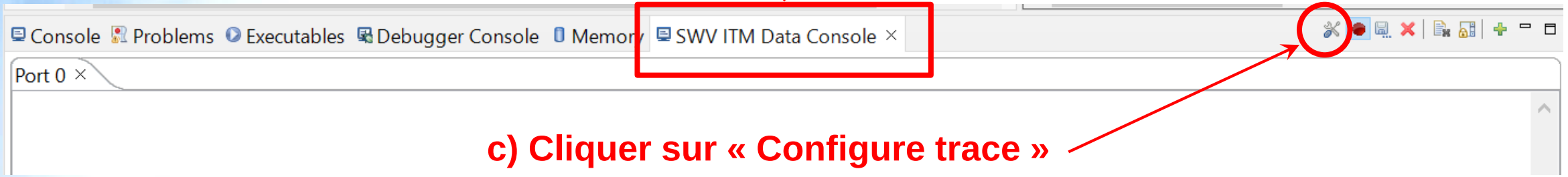
8. Console de Données

10 **Ouvrir Windows > Show View > SWV > SWV ITM Data Console:**

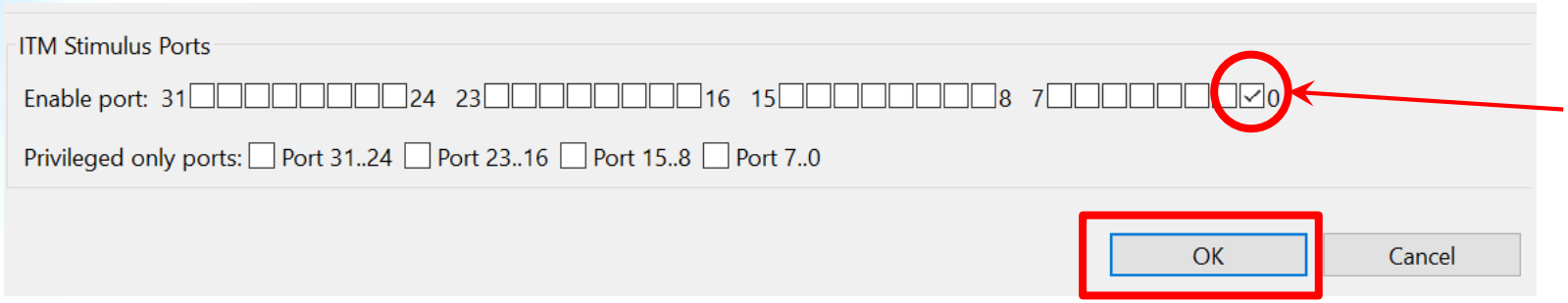


La console de données SWV ITM (Instrumentation Trace Macrocell) est un outil utilisé pour afficher ces données de trace lors des sessions de débogage, fournissant des informations sur le comportement du logiciel embarqué s'exécutant sur le processeur Cortex-M

b) Un nouvel onglet « SWV ITM Data Console » s'ajoute Ici



c) Cliquer sur « Configure trace »



d) Cocher Port 0

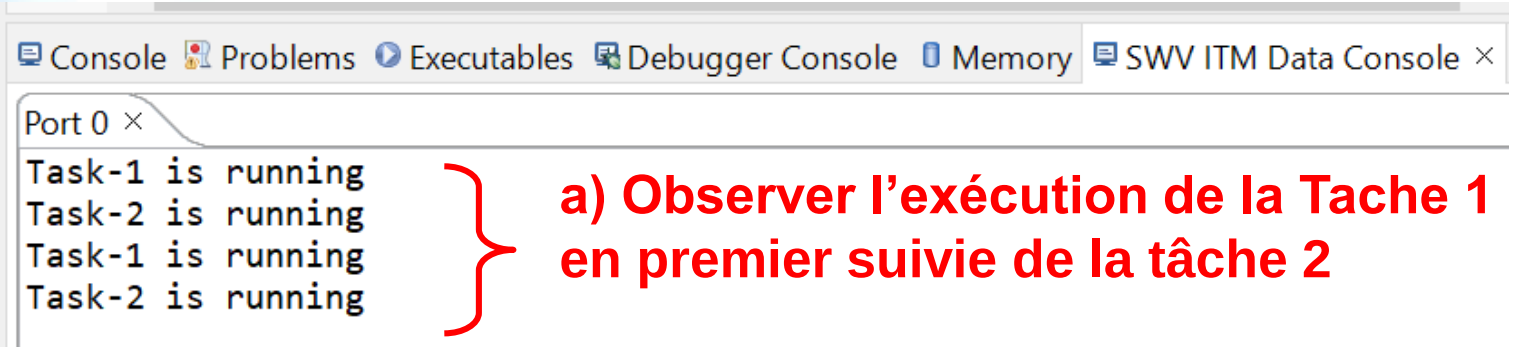
e) Cliquer sur OK

Remarque: Le port 0 c'est celui qu'on a utilisé dans le code ajouté au fichier syscalls.c

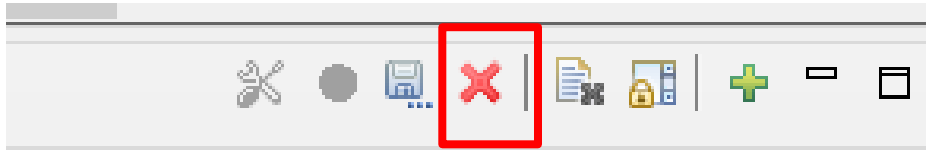
9. Exécution et tests

11 **Cliquer, enfin, sur ce bouton RESUME en VERT :**

Cela permet d'exécuter la suite du programme.



a) Observer l'exécution de la Tache 1 en premier suivie de la tâche 2



b) Cliquer sur (x) « Remove all … » et observer de nouveau l'affichage!

10. Exercice d'application

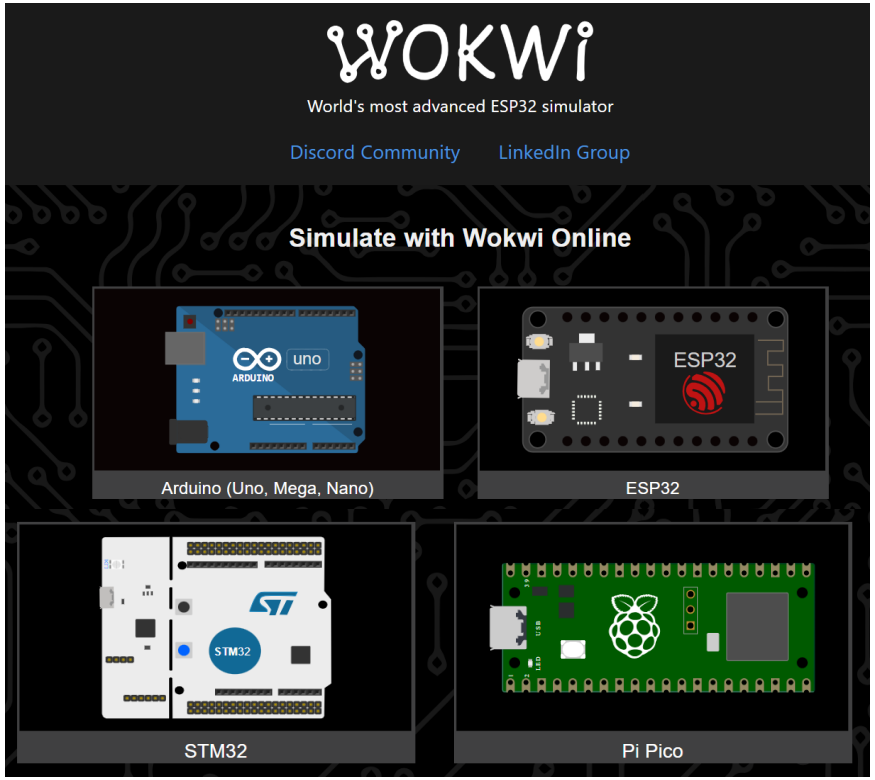
1) Permuter les priorité des deux tâches:
   - La Tâche 2 de priorité supérieure à celle de la Tâche 1.
   - Observer les changements au niveau de l'exécution.

2) Avec la priorité initiale (T1 plus prioritaire que T2), on souhaite inclure dans le code de la Tâche 2 une ligne qui termine l'exécution de la Tâche 1:
   - Utiliser: **osThreadTerminate(Task1Handle);**
   - Observer de nouveau les changements au niveau de l'exécution.

III- Utilisation de FreeRTOS avec la les plateformes de Simulation

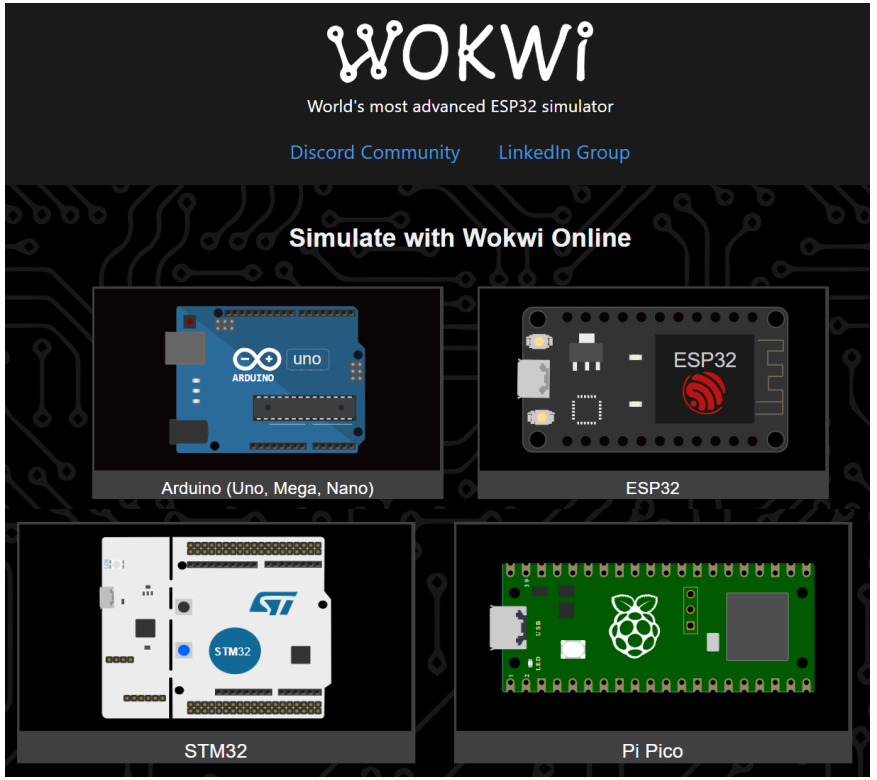
1. Wokwi

- Simulation facile de l'ESP32, supportant FreeRTOS (intégré nativement dans l'ESP32).
- Interface utilisateur simple avec éditeur de code intégré.
- Simulation de périphériques comme LEDs, boutons, capteurs, etc.



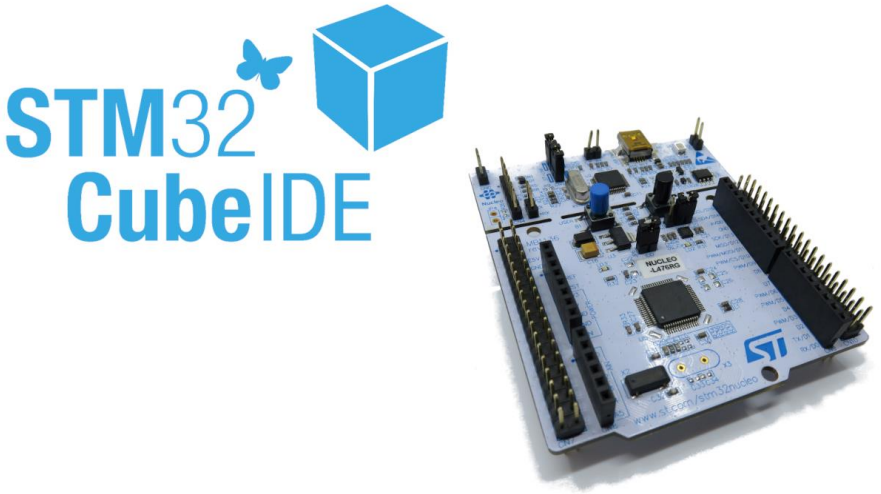
... Wokwi

```
#include <Arduino.h>

void Task1(void *pvParameters) {
    while (1) {
        Serial.println("Task 1 is running...");
        vTaskDelay(pdMS_TO_TICKS(1000)); // Pause de 1 seconde
    }
}

void Task2(void *pvParameters) {
    while (1) {
        Serial.println("Task 2 is running...");
        vTaskDelay(pdMS_TO_TICKS(500)); // Pause de 500 ms
    }
}

void setup() {
    Serial.begin(115200);

    // Création des tâches
    xTaskCreate(Task1, "Task 1", 1024, NULL, 1, NULL);
    xTaskCreate(Task2, "Task 2", 1024, NULL, 1, NULL);
}

void loop() {
    // Rien dans loop, tout est géré par FreeRTOS
}
```



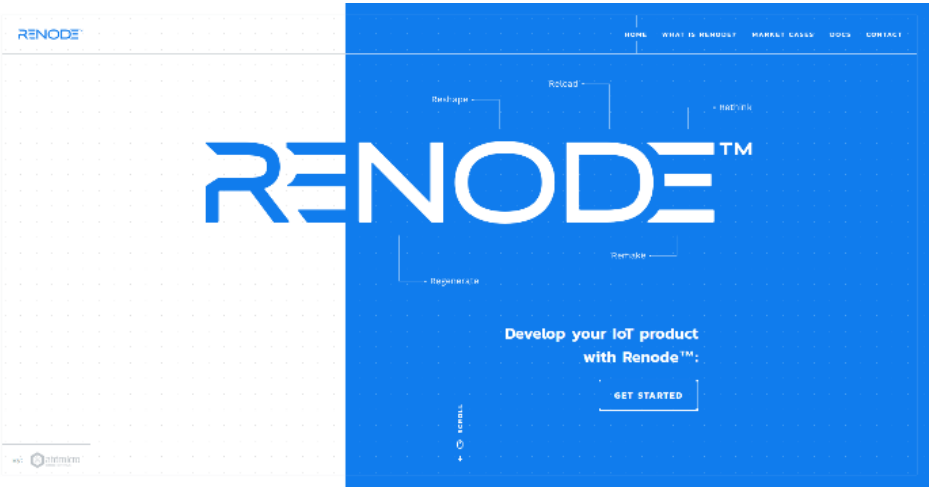
2. STM32CubeIDE + QEMU (simulateur open source)

- **Support** : STM32 uniquement.
- **Fonctionnalités** :
  - Simulation matérielle basique pour les cartes STM32.
  - Compatible avec FreeRTOS via l'intégration dans STM32CubeIDE.
  - QEMU est utilisé pour émuler les architectures ARM Cortex-M (par exemple, STM32).
- **Limitation** : La simulation est limitée à l'architecture et aux périphériques de base (GPIO, UART).
- **Installation nécessaire** : QEMU et STM32CubeIDE.
- **Lien pour QEMU** : https://www.qemu.org



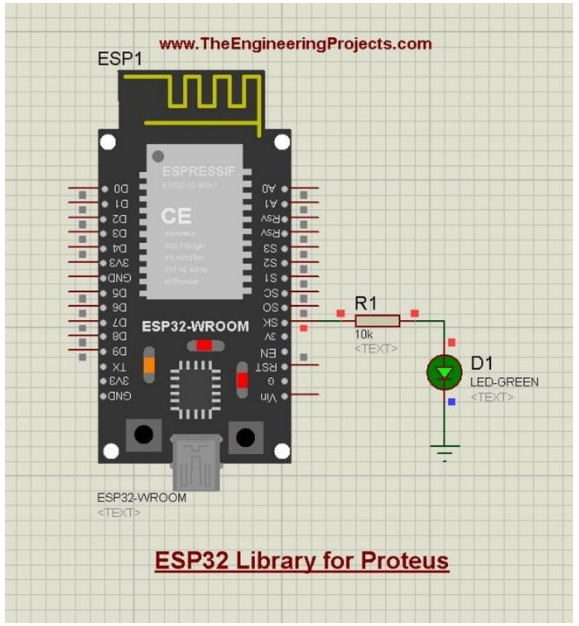
3. Renode

- **Support** : ESP32, STM32, et autres architectures ARM.
- **Fonctionnalités** :
  - Simulation avancée pour les systèmes embarqués avec support de FreeRTOS.
  - Permet de tester les interactions entre plusieurs nœuds (réseaux IoT, capteurs).
  - Simulation précise des périphériques matériels.
- Lien: https://renode.io



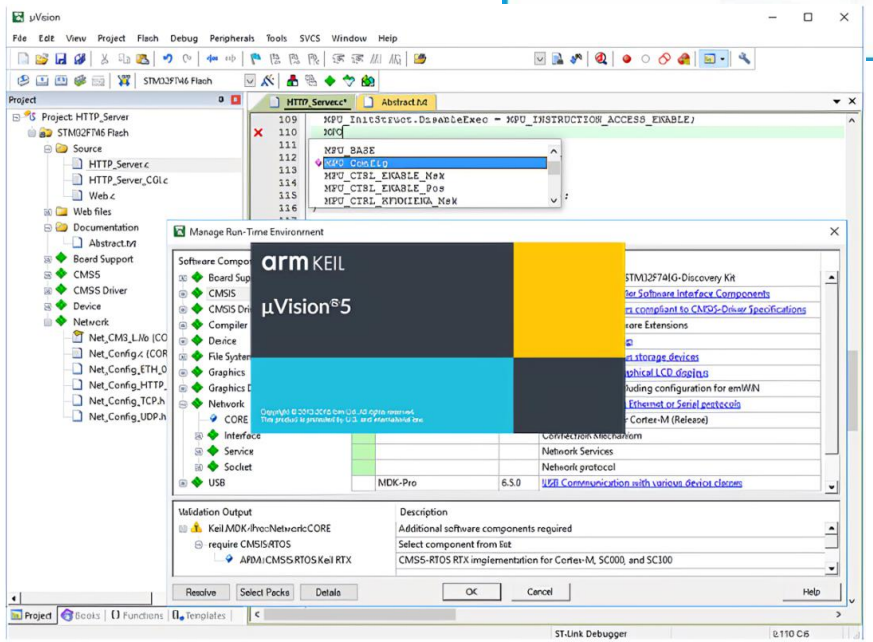
4. Proteus Design Suite (avec extensions)

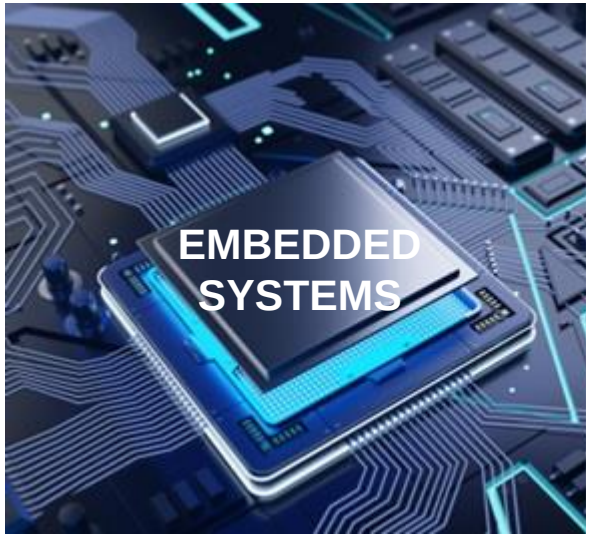
- **Support** : ESP32 (avec librairies Arduino), STM32 (partiellement supporté via ARM Cortex-M).
- **Fonctionnalités** :
  - Simulation graphique des microcontrôleurs et circuits complets.
  - Permet de tester du code embarqué (FreeRTOS possible avec des réglages).
  - Support pour UART, I2C, SPI et d'autres protocoles courants.
- **Limitation** : Logiciel propriétaire, payant après une période d'essai.
- Lien: https://www.labcenter.com



5. ARM Keil MDK + µVision Debugger

- **Support** : STM32.
- **Fonctionnalités** :
  - Simulation de microcontrôleurs ARM Cortex-M avec FreeRTOS.
  - Analyse en temps réel avec RTOS Event Viewer.
- **Limitation** : Simulation plus axée sur le débogage du code que sur les interactions matérielles.
- Lien: https://www.keil.com





FIN !
Questions ?