FreeRTOS

FreeRTOS est un système d'exploitation temps-réel sous licence MIT et ayant été racheté par Amazon en 2017.

Le site de FreeRTOS est très complet et propose plusieurs documentations différentes (https://www.freertos.org/RTOS.html) :

- **Developper Docs :** de la documentation plutôt généraliste (applicable à d'autres OS) sur les concepts et mécanismes liés aux RTOS. Considérez cette partie comme un cours rapide sur les RTOS.
- **Secondary Docs :** de la documentation un peu plus poussée, et en même temps un peu plus centrée sur FreeRTOS.
- **API Reference :** la documentation sur l'API (les fonctions et services) proposés par FreeRTOS.

I. Porter FreeRTOS dans un projet

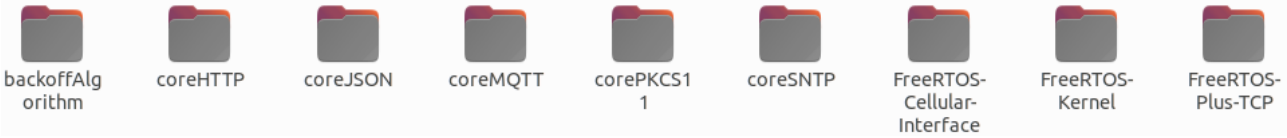
I.1. Sources et arborescence

Les sources sont accessibles directement par le site de FreeRTOS :

https://www.freertos.org/a00104.html

Le site propose généralement deux versions en téléchargement : la dernière ***release*** en date et la version **LTS (*Long Term Support*)**. La dernière *release* a pour avantage de contenir les nouveautés de l'OS mais n'est maintenue que jusqu'à la sortie de la version suivante. La version LTS est potentiellement plus ancienne (de quelques mois) mais le support (corrections de bugs critiques et sécurité) est garanti pendant deux ans.

Dans les deux cas, la version téléchargée contient non seulement les sources du *kernel*, mais également des sources AWS pour l'IoT (suite au rachat par Amazon) et d'autres librairies (HTTP, TCP, …).



La même page Web propose aussi une redirection vers GitHub, qui offre la possibilité de ne télécharger que les sources du *kernel* (https://github.com/FreeRTOS/FreeRTOS-LTS). C'est ceci qui a été téléchargé et fourni dans l'archive de TP.

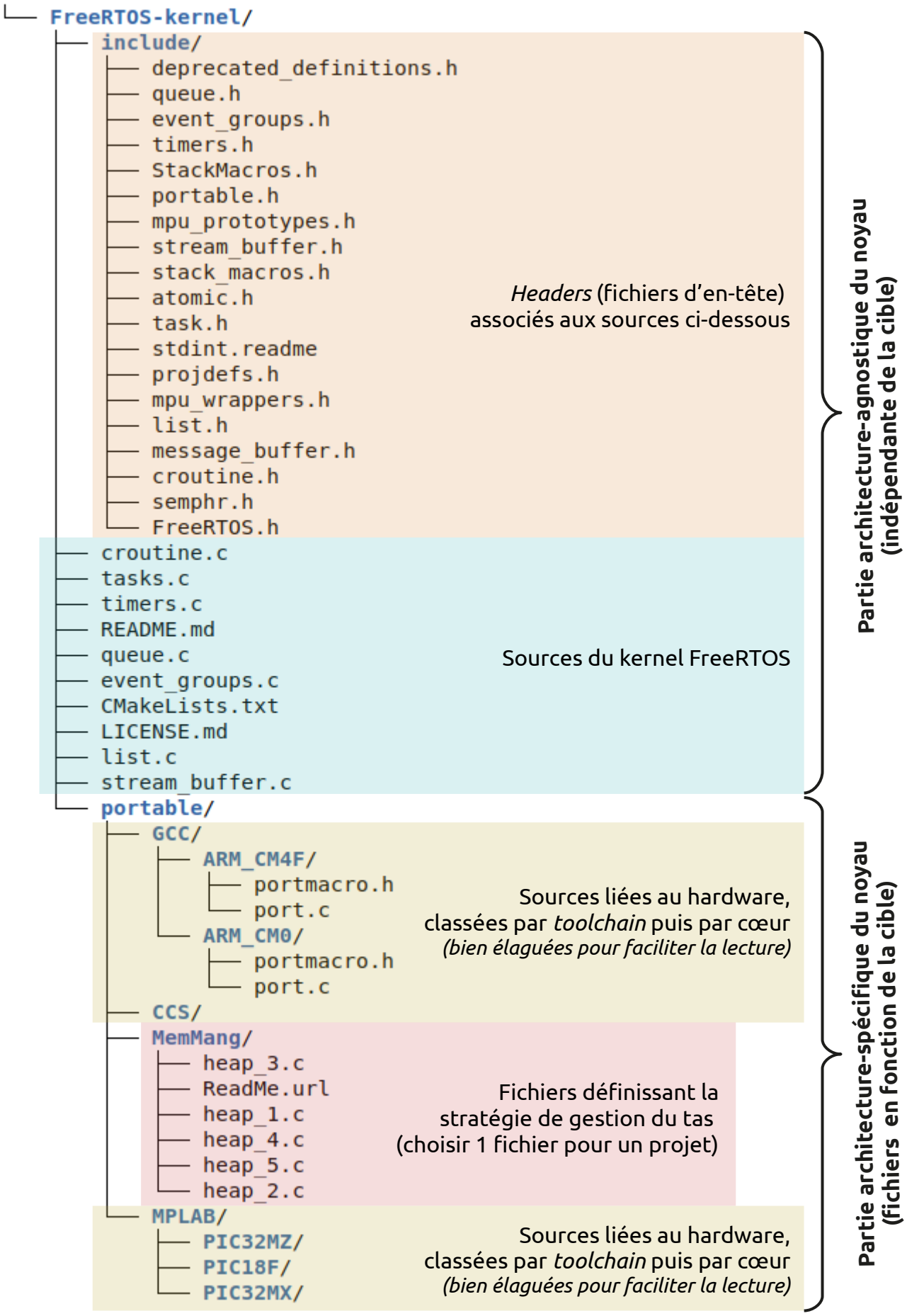
Si on se concentre uniquement sur les sources du *kernel*, l'arborescence est donnée sur la page suivante. On y distingue deux ensembles de fichiers.

Les **fichiers architecture-agnostiques** (situés à la racine et dans `include/`) sont génériques et utilisés quelle que soit l'architecture du MCU cible. Certains fichiers peuvent ne pas être intégrés au projet tant que les fonctionnalités ne sont pas utilisées (par exemple `queue.c`, …) .

Les **fichiers architecture-spécifiques** (situés dans `portable/`) sont eux propres à l'architecture cible. Certains fichiers sont mêmes en assembleur. Ces fichiers doivent être intégrés au projet car ils contiennent les définissions de fonctions appelées par les sources génériques (comme par exemple la configuration du *timer* pour l'ordonnanceur). Ce répertoire contient des sources pour plus de 40 architectures différentes[1] (la représentation page suivante a été allégée pour des raisons de lisibilité). Les sources sont classées par *toolchain* (`GCC` dans notre cas), puis par architecture cible (`ARM_CM0` pour Cortex-M0).

Toujours dans `portable/`, le répertoire `MemMang/` **(*Memory Management*)** ne correspond pas à une *toolchain* mais contient 5 fichiers correspondant chacun à une stratégie de gestion du tas (allocation, désallocation, ...). Un de ces fichiers doit être choisi et intégré au projet.

1 https://freertos.org/RTOS_ports.html

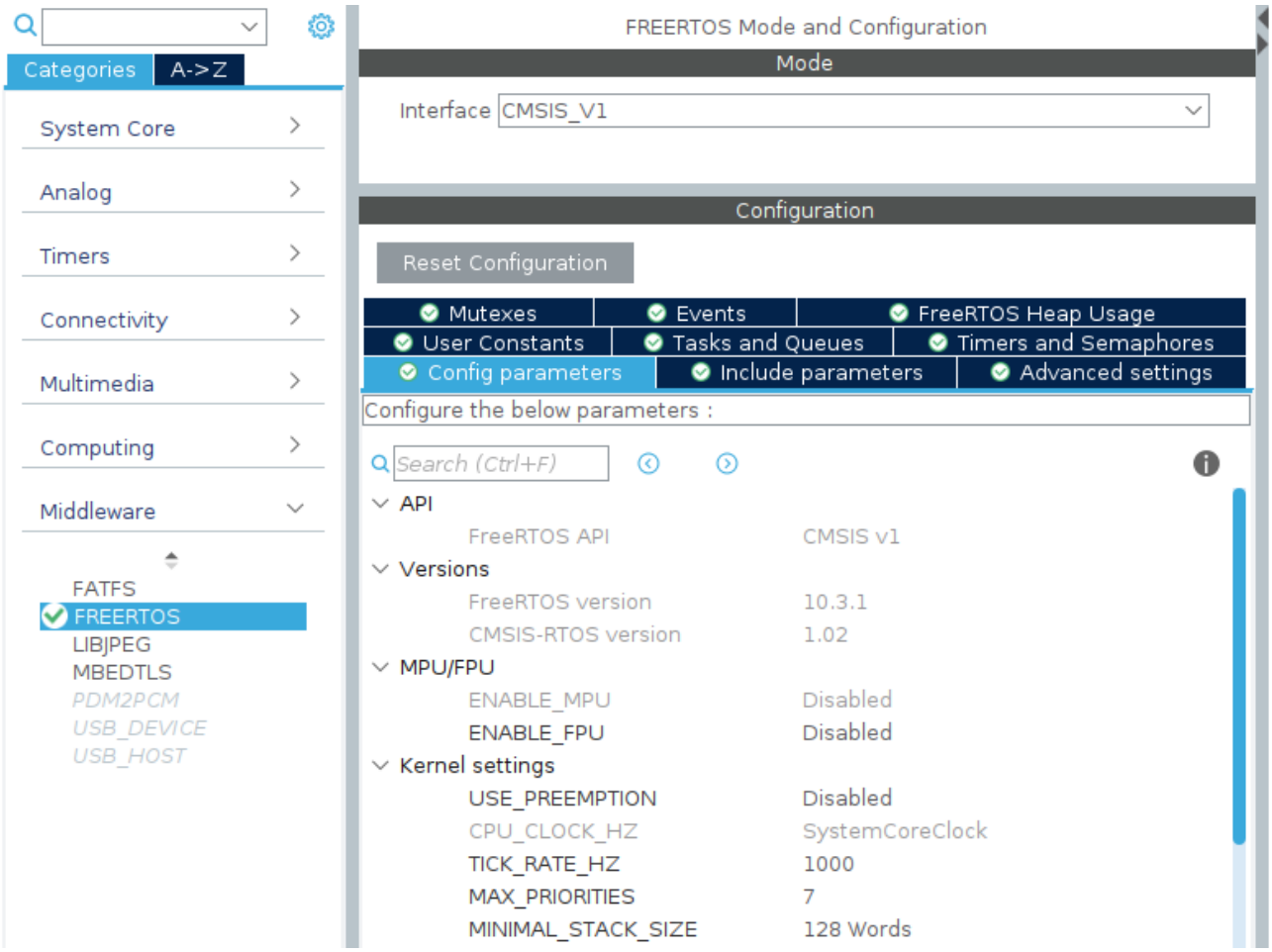

FreeRTOS-kernel/
include/
deprecated_definitions.h
queue.h
event_groups.h
timers.h
StackMacros.h
portable.h
mpu_prototypes.h
stream_buffer.h
stack_macros.h
atomic.h
task.h
stdint.readme
projdefs.h
mpu_wrappers.h
list.h
message_buffer.h
croutine.h
semphr.h
FreeRTOS.h
croutine.c
tasks.c
timers.c
README.md
queue.c
event_groups.c
CMakeLists.txt
LICENSE.md
list.c
stream_buffer.c
portable/
GCC/
ARM_CM4F/
portmacro.h
port.c
ARM_CM0/
portmacro.h
port.c
CCS/
MemMang/
heap_3.c
ReadMe.url
heap_1.c
heap_4.c
heap_5.c
heap_2.c
MPLAB/
PIC32MZ/
PIC18F/
PIC32MX/
Headers (fichiers d'en-tête) associés aux sources ci-dessous
Sources du kernel FreeRTOS
Sources liées au hardware, classées par toolchain puis par cœur (bien élaguées pour faciliter la lecture)
Fichiers définissant la stratégie de gestion du tas (choisir 1 fichier pour un projet)
Sources liées au hardware, classées par toolchain puis par cœur (bien élaguées pour faciliter la lecture)
Partie architecture-agnostique du noyau (indépendante de la cible)
Partie architecture-spécifique du noyau (fichiers en fonction de la cible)

I.2. Ajouter FreeRTOS depuis STM32CubeMX

FreeRTOS peut directement être intégré à un projet STM32Cube grâce à l'outil de configuration SMT32CubeMX, soit lors de la création du projet, soit en ouvrant le fichier `*.ioc` associé et en re-générant le code.

Les champs situés dans les onglets « *Config parameters* » et « *Include parameters* » correspondent aux définitions des macro-constantes du fichier `FreeRTOSConfig.h`. Ces macros restent directement éditables dans le fichier, mais elles seront redéfinies en cas de re-génération du code par CubeMX.

De la même manière les tâches, sémaphores, mutex et autres peuvent directement être créés depuis cette fenêtre de configuration. Toutefois, il faut se rappeler que toute configuration avec CubeMX sera réalisée par un *wrapping* de l'API FreeRTOS par l'API CMSIS-RTOS (v1 ou v2).



Bref : *clic & select* → *Generate code* → Ding, c'est prêt !

Vous remarquerez la création d'un répertoire `Middlewares/Third_Party/FreeRTOS/` qui contient les sources du kernel de FreeRTOS, désormais inclus au projet.

I.3. Ajouter manuellement les fichiers à un projet

Il est également possible d'intégrer à un projet les sources de FreeRTOS sans passer par CubeMX. Pour savoir quels fichiers de `FreeRTOS-Kernel/` doivent être intégrés au projet, poursuivez la lecture de cette section. Pour savoir comment les intégrer au projet, reprenez la section « Error: Reference source not found Error: Reference source not found » de l'annexe.

Parmi les fichiers des sources (à la racine de `FreeRTOS-Kernel/`), seuls les fichiers C utilisés peuvent être intégrés au projet. Les fichiers dans `include/` doivent être importés (avec les chemins d'inclusion précisés à la *toolchain*).

Parmi les fichiers du répertoire `portable/`, il ne faut sélectionner que le répertoire correspondant à la fois à la toolchain et au processeur cible. Dans notre cas, il s'agit respectivement de `GCC` (GNU's Compiler Collection) et `ARM_CM0` (ARM Cortex-M0). Les autres répertoires ne doivent pas être intégrés au projet (sauf point suivant).

Enfin, un fichier de `portable/MemMang/` doit obligatoirement être intégré au projet, suivant la stratégie que l'on souhaite employer pour la gestion du tas.

En bref :

- `FreeRTOS-Kernel/`
  - fichiers C : prendre uniquement les fichiers nécessaires
  - `include/` : tout inclure
  - `portable/MemMang/` : prendre uniquement le `heapX.c` que l'on souhaite
  - `portable/GCC/ARM_CM0/` : tout inclure
  - `portable/` : ne prendre aucun des autres répertoires

II. Fichier FreeRTOSConfig.h

Le fichier `FreeRTOSConfig.h` est un fichier ne comportant que des macro (pas de déclaration de fonction). Il s'agit du seul fichier physiquement sorti du système de fichiers de FreeRTOS, et est par défaut intégré avec les fichiers liés à l'application. Ce fichier est essentiel et assure la configuration du noyau avant compilation.

Par exemple, de nombreuses macros sont préfixées par `config`. Il s'agit donc des constantes permettant de configurer FreeRTOS dans un mode de fonctionnement propre à l'application. On peut par exemple choisir le mode de l'ordonnanceur (coopératif ou préemptif), la fréquence d'interruption de l'ordonnanceur, les modes d'allocations, la taille du tas ou des piles, … Autant de paramètres obligatoires mais ajustables.

Un autre ensemble de macros, celles préfixées par `INCLUDE_`, permet de sélectionner les fonctionnalités intégrées (ou non) à FreeRTOS. Mettre les macros à '0' permet donc de désactiver certaines fonctionnalités (fonctions de l'API), même si les fonctionnalités élémentaires (création de tâches, démarrage du *scheduler*, …) sont toujours présentes. L'idée est ne compiler, linker et écrire dans la mémoire programme du processeur que les fonctions utilisées dans l'optique de réduire à la fois le temps de compilation et la taille du programme binaire.

```
/*
 * FreeRTOS Kernel V10.3.1
 * [...]
 * See http://www.freertos.org/a00110.html
 *----------------------------------------------------------*/

#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H

#define configENABLE_FPU                        0
#define configENABLE_MPU                        0

#define configUSE_PREEMPTION                    0
#define configSUPPORT_STATIC_ALLOCATION         0
#define configSUPPORT_DYNAMIC_ALLOCATION        1
#define configUSE_IDLE_HOOK                     1
#define configUSE_TICK_HOOK                     0
#define configCPU_CLOCK_HZ                      ( SystemCoreClock )
#define configTICK_RATE_HZ                      ((TickType_t)1000)
#define configMAX_PRIORITIES                    ( 7 )
#define configMINIMAL_STACK_SIZE                ((uint16_t)128)
#define configTOTAL_HEAP_SIZE                   ((size_t)15360)
#define configMAX_TASK_NAME_LEN                 ( 16 )
#define configUSE_16_BIT_TICKS                  0
#define configIDLE_SHOULD_YIELD                 0
#define configQUEUE_REGISTRY_SIZE               8
#define configENABLE_BACKWARD_COMPATIBILITY     0
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES                   0
#define configMAX_CO_ROUTINE_PRIORITIES         ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */
#define INCLUDE_vTaskPrioritySet            1
#define INCLUDE_uxTaskPriorityGet           0
#define INCLUDE_vTaskDelete                 1
#define INCLUDE_vTaskCleanUpResources       0
#define INCLUDE_vTaskSuspend                0
#define INCLUDE_vTaskDelayUntil             0
#define INCLUDE_vTaskDelay                  1
#define INCLUDE_xTaskGetSchedulerState      1
#define INCLUDE_xTaskResumeFromISR          0


#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 5
#define configCHECK_FOR_STACK_OVERFLOW               2
#define configUSE_MALLOC_FAILED_HOOK                 1

#endif /* FREERTOS_CONFIG_H */
```