TP électronique

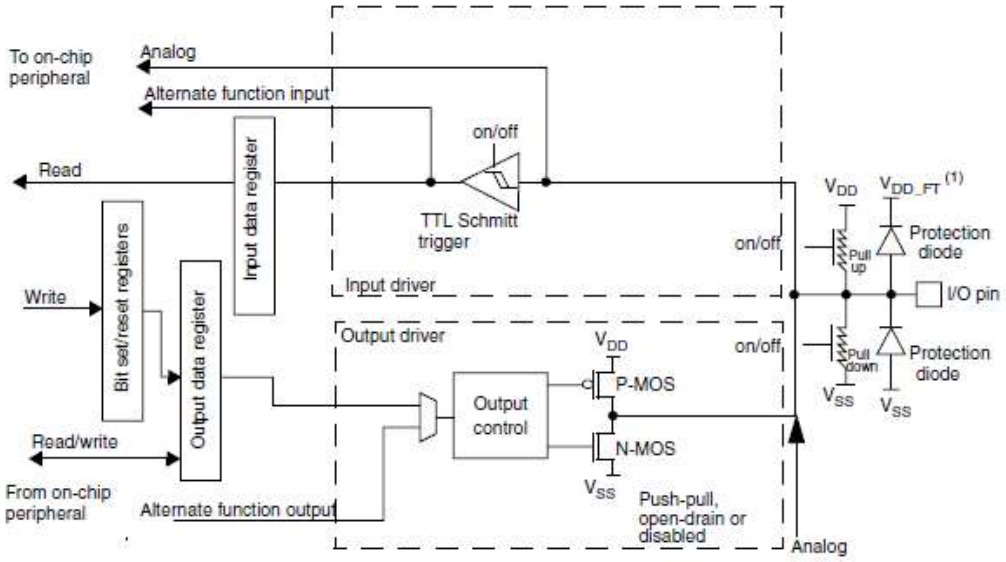
Microcontrôleur STM32L152

GPIO

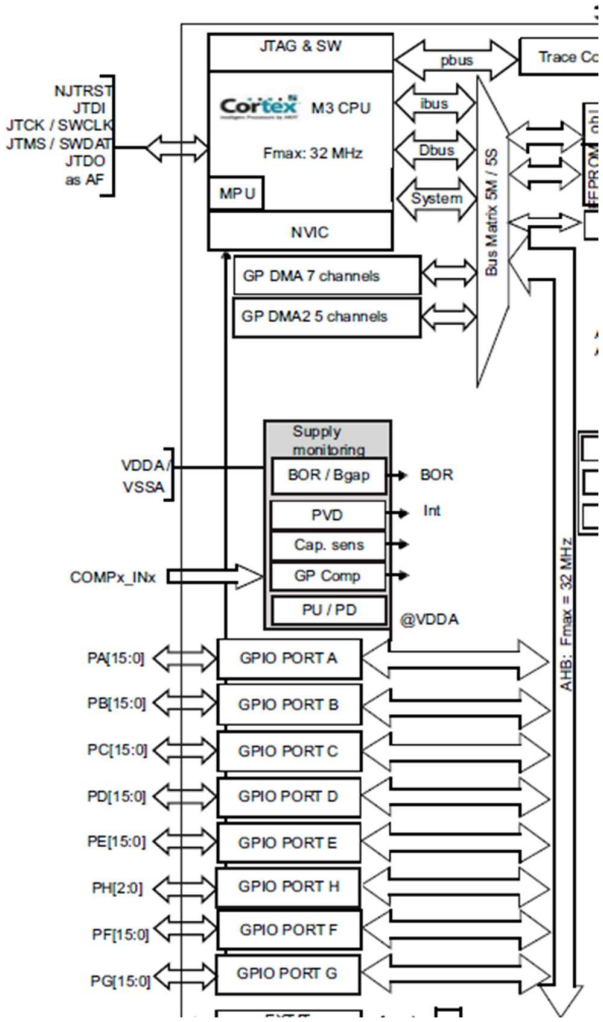
Chapitre 1 GPIO Introduction

GPIO signifie General Purpose Input Output. Les broches peuvent avoir plusieurs fonctions, mais hormis les broches réservées pour fournir la tension ou éventuellement une oscillation, elles peuvent toutes être connectées à un port. Ceci permet de connecter le monde extérieur au microcontrôleur.

Le schéma global d'un port est dessiné ci-dessous.

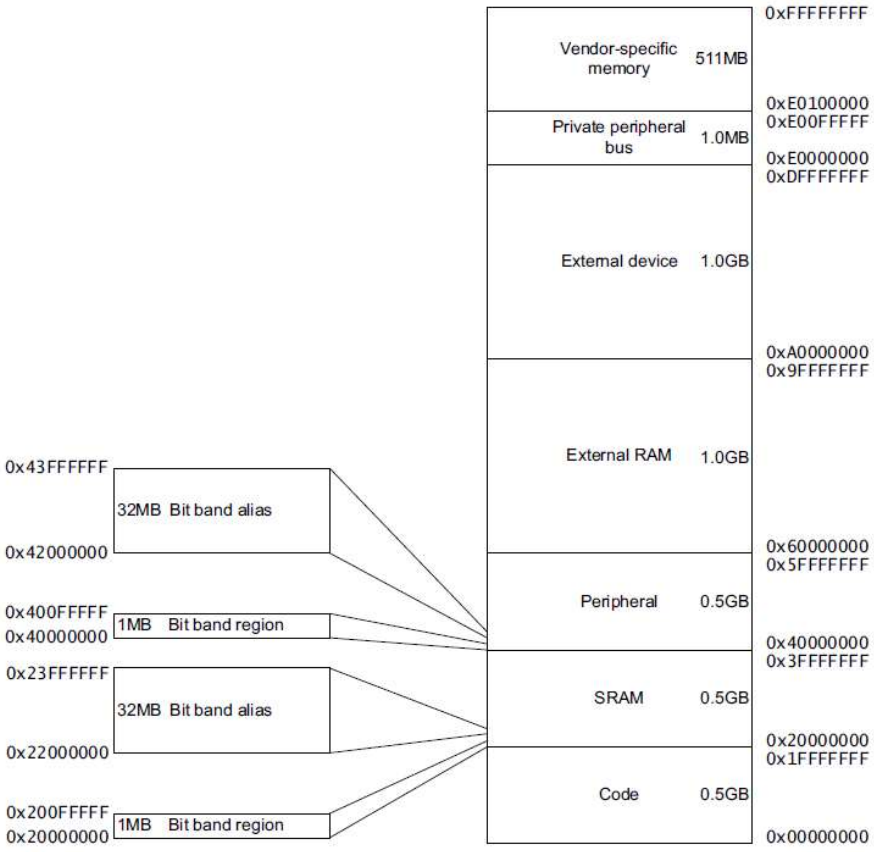


Dans le STM32L152RE, l'accès aux bits de configuration du port se fait via le bus interne AHB



Attention, ceci n'est pas le cas dans tous les cortex M, principalement les nouveaux microcontrôleurs qui ont un bus dédié pour les ports de GPIO.

Les GPIOs sont considérés comme des périphériques et les registres de configurations sont donc localisés dans le block d'adresses 0x4000 0000



Question :

Donner les adresses des ports A, B, C, D et E. Donner les noms des registres qui composent un port et leurs fonctions respectives.

Chapitre 2. Codage en C d'un programme en Assembleur

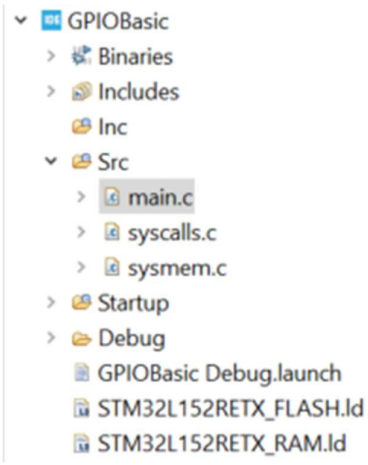
Voici un code assembleur écrit pour le STM32l152RE.

```
/* Enable port A */
ldr r0, =0x4002381C     /* address of the RCC_AHBENR GPIO Clock Enable Register */
ldr r1, =0x00000001     /* mask to apply to RCC_AHBENR to Enable GPIOAEN */
ldr r2, [r0]            /* load value of RCC_AHBENR to r2 */
orr r1,r2               /* for GPIOEN to 1 to enable port A */
str r1, [r0]            /* write new value to RCC_AHBENR */

/* configure pin in output mode */
ldr r0, =0x40020000     /* address of the MODE Register port A */
ldr r1, =0x00000400     /* mask to apply to MODE Register port A */
ldr r2, [r0]            /* load value of MODE Register port A  */
orr r1,r2               /* force bit 10 to 1  */
str r1, [r0]            /* write new value to port A MODER */

ldr r1, =0xFFFFF7FF     /* mask to apply to MODE Register port A */
ldr r2, [r0]            /* load value of MODE Register port A  */
and r1,r2               /* force bit 11 to 0  */
str r1, [r0]            /* write new value to port A MODE R */

ldr r0, =0x40020014     // address of port A ODR register
ldr r1, =0x00000020     /* mask to apply to port A ODR */
ldr r2, [r0]            /* load value of port A ODR */
orr r1,r2               /* for port A ODR to 1 to to put this pin to high level */
str r1, [r0]            /* write new value to port A ODR */
```

Question :
Comprendre ce code et l'écrire en langage C
Pour AHB, on utilisera l'adresse directement et pour le GPIO on utilisera une structure.
Pour ce ceci vous devez créer un nouveau projet.

ATTENTION !!!!!!!!!!!!!!!!!!
Créer les projets de manière identique à la création de projets en assembleur. Mais NE PAS REMPLACER LE FICHIER DE START UP. Conserver le fichier créé par le STM32CubeIDE et qui permet d'appeler le fichier main.c

Pour écrire le code C, aller dans le fichier main.c



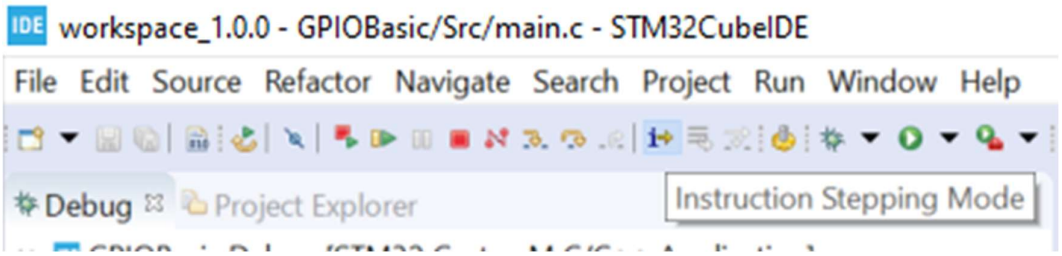
La fonction main et la boucle infinie sont codées dans le fichier main .c

```
/**
  ******************************************************************************
  * @file           : main.c
  * @author         : Auto-generated by STM32CubeIDE
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */

#if !defined(__SOFT_FP__) && defined(__ARM_FP)
  #warning "FPU is not initialized, but the project is compiling for an FPU. Please initialize the FPU before use."
#endif

int main(void)
{

    for(;;);
}
```

Ecrire le code dans ce fichier.

<u>Remarque pour fenêtre de désassemblage :</u>

Lorsqu'un code est écrit en C, on peut le faire avancer instruction par instruction de C ou par instruction par instruction de langage assembleur. Le choix se faire par l'icône « Instruction Stepping Mode ». Si il est appuyé, chaque appui sur « step into » avance d'une instruction an assembleur. Dans le cas contraire, « step into » fait avancer d'une instruction en C.



<u>Question :</u>
Vérifier le code assembleur généré par le compilateur.
Quelle est la différence dans les instructions avec le code donné au début de ce chapitre ?

Chapitre 3 Autre moyen de contrôler un GPIO

La sortie d'un GPIO sur le STM32L152RE peut être contrôlée de différentes manières :

- En utilisant le registre ODR
- En utilisant le registre BSRR

Pour étudier le comportement du registre BSSR, reprendre le projet créé au chapitre 2

Questions :

- ✓ Trouver l'adresse du registre BSRR qui contrôle la broche PA5
- ✓ Ecrire directement dans ce registre en utilisant les pointeurs pour allumer la ED ou éteindre la LED.
- ✓ Ecrire le code en utilisant les structures
- ✓ Faire un programme qui fait clignoter la LED en utilisant cette tempo

```
for (uint32_t i = 0; i < 200000; i++) {
}
```

- ✓ Grâce à la fenêtre de désassemblage, vérifier le codage en assembleur lorsque le registre BSRR est utilisé par rapport à l'utilisation du registre ODR.

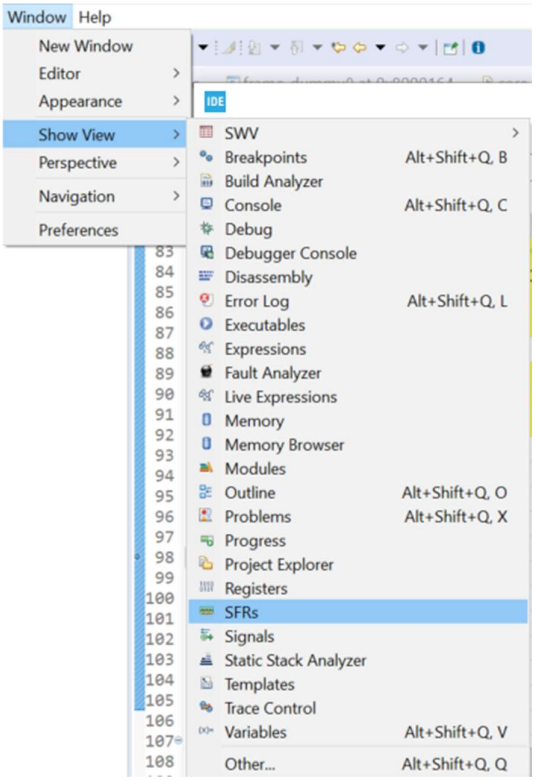
L'avantage du BSSR par rapport à l'ODR est l'inutilité de faire un masque. En effet, quand le registre ODR est utilisé, les étapes suivantes sont nécessaire :

1. Lecture du registre ODR
2. Génération du masque
3. Calcul logique entre l'ODR et le masque de la nouvelle valeur à écrire dans l'ODR
4. Ecriture dans l'ODR

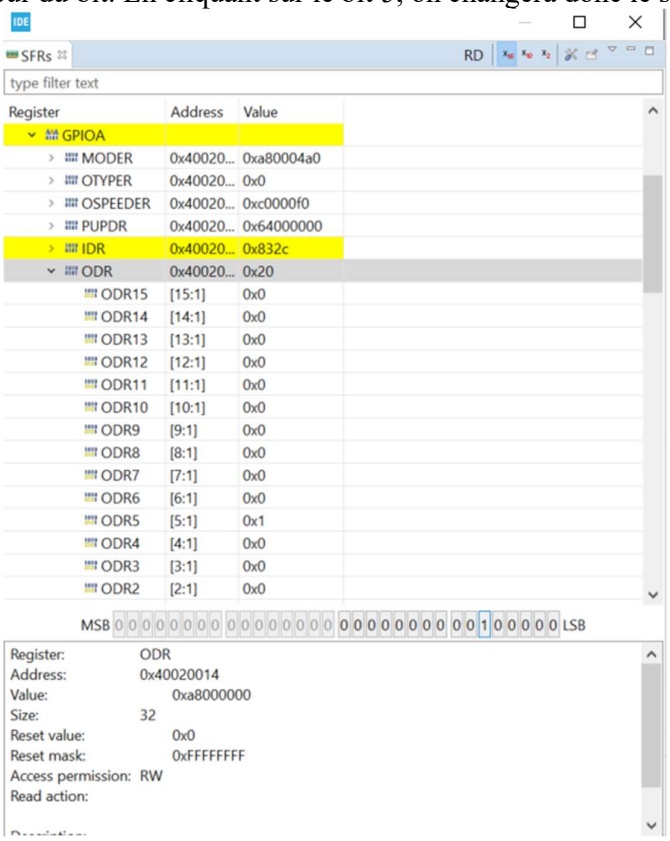
Avec le BSSR, Moins d'étapes sont nécessaires :

1. Génération du masque
2. Ecriture dans le BSSR

En mode debug, quand le programme est en pause, il est également possible de modifier les registres via le STM32CubeIDE comme montré ci-dessous.
Ouvrir la fenêtre SFR (Special Function Register)



Sélectionner GPIOA et développer les différents sous-menus. Nous pouvons avoir accès à chaque bit de chaque registre. En cliquant sur le bon bit de la barre de MSB à LSB, on peut faire changer la valeur du bit. En cliquant sur le bit 5, on changera donc le statut de la LED



Question :
Grâce aux informations du system viewer, retrouver les adresses précédentes.

STMicroelectronics fournit du code pour simplifier et accélérer le travail du développeur.

Question :
Créer un nouveau projet suivant les indications de l'annexe : Créer un projet stm32 avec Cube MX.

Chaque périphérique est relié à une structure. Précédemment, nous avons écrit notre propre structure. Ceci est un travail fastidieux qui a déjà été fait par le fournisseur de microcontrôleurs . Par exemple, l'accès à une GPIO est fait grâce à la structure suivante.qui se trouve dans le fichier stm32l152xe.h


```
/**
  * @brief General Purpose IO
  */

typedef struct
{
  __IO uint32_t MODER;        /*!< GPIO port mode register,                  Address offset: 0x00      */
  __IO uint32_t OTYPER;       /*!< GPIO port output type register,           Address offset: 0x04      */
  __IO uint32_t OSPEEDR;      /*!< GPIO port output speed register,          Address offset: 0x08      */
  __IO uint32_t PUPDR;        /*!< GPIO port pull-up/pull-down register,     Address offset: 0x0C      */
  __IO uint32_t IDR;          /*!< GPIO port input data register,            Address offset: 0x10      */
  __IO uint32_t ODR;          /*!< GPIO port output data register,           Address offset: 0x14      */
  __IO uint32_t BSRR;         /*!< GPIO port bit set/reset registerBSRR,     Address offset: 0x18      */
  __IO uint32_t LCKR;         /*!< GPIO port configuration lock register,    Address offset: 0x1C      */
  __IO uint32_t AFR[2];       /*!< GPIO alternate function register,         Address offset: 0x20-0x24 */
  __IO uint32_t BRR;          /*!< GPIO bit reset register,                  Address offset: 0x28      */
} GPIO_TypeDef;
```

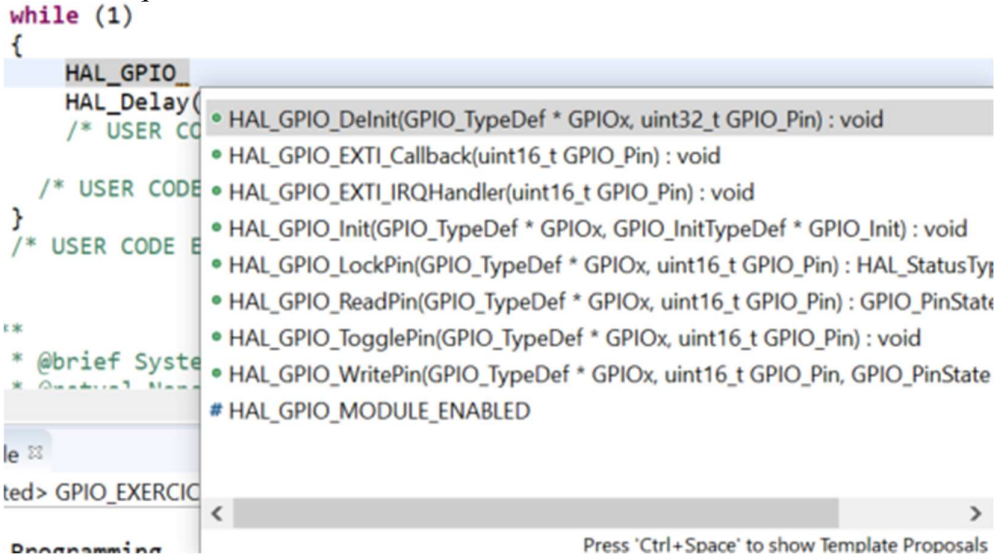
Question :
Dans ce nouveau projet, réécrire le programme du clignotement de la LED en utilisant les structures fournies par STMicroelectronics (sans faire référence à la structure défnie par vos soins dans les exercices précédents)..
Vous voyez ainsi que cette programmation est très rapide.

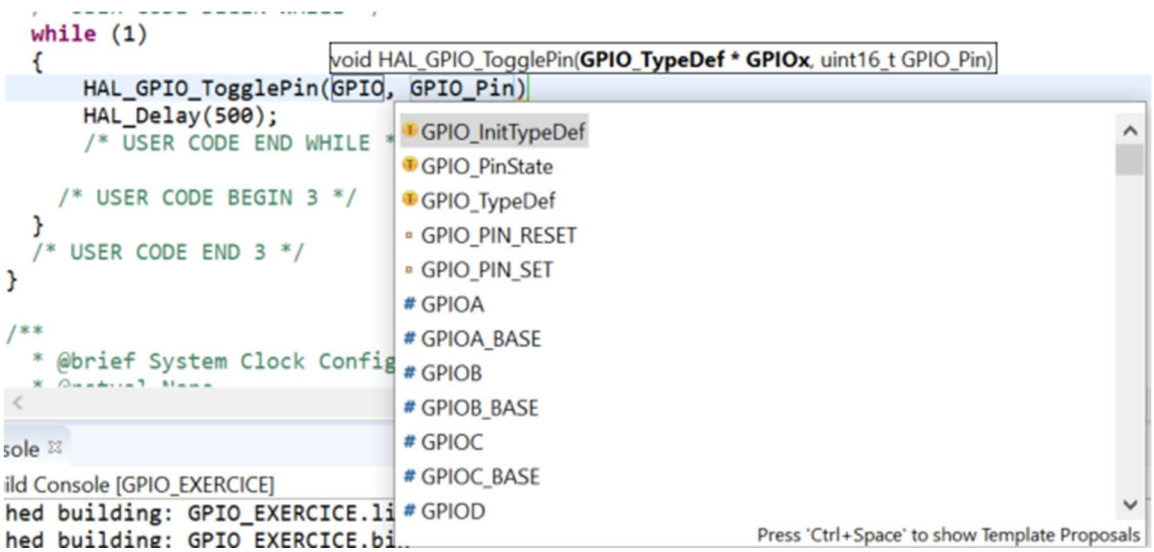
Au-delà de ces structures, afin de simplifier encore plus le développement d'un programme, STMicroelectronics fournit un middleware appelé HAL Library. HAL signifie Hardware Abstract Layer. L'objectif est d'écrire du code qui est indépendant de la plateforme matérielle, et donc du code réutilisable. Ce sont les header files spécifiques pour chaque plateforme qui permettront de compiler un code utilisable pour chaque plateforme.

Dans le main.c, écrire le code ci-dessous

```
/* USER CODE BEGIN WHILE */
while (1)
{
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    HAL_Delay(500);
    /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

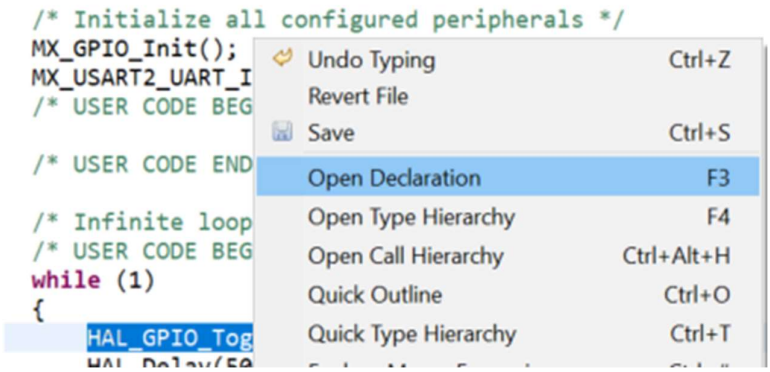
La complétion automatique aide à écrire la code très rapidement et à éviter les fautes d'orthographe.
Deux exemples ci-dessous.





Compiler le programmer et le lancer en mode debug. La LED clignote avec une pause de 500ms entre chaque état.

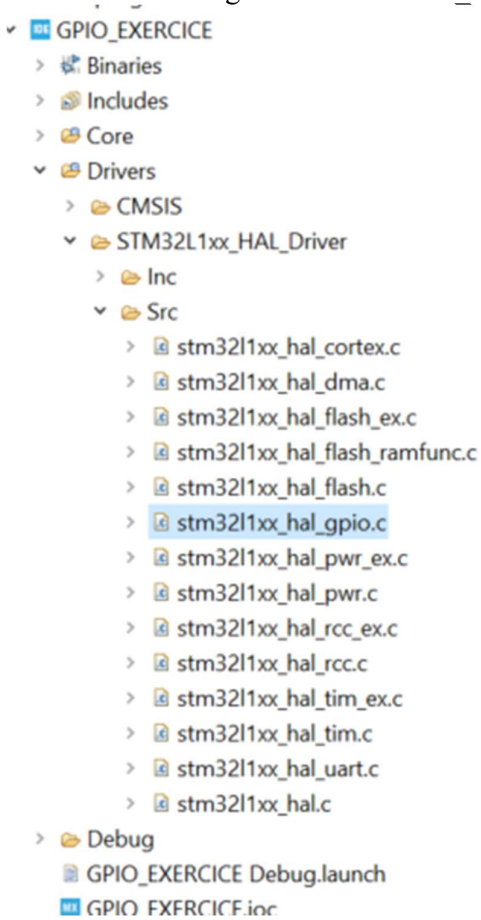
Il est possible de voir comment le code d'une fonction en surlignant la fonction, puis en cliquant sur le bouton droit de la souris et choisir Open Declaration dans le menu.



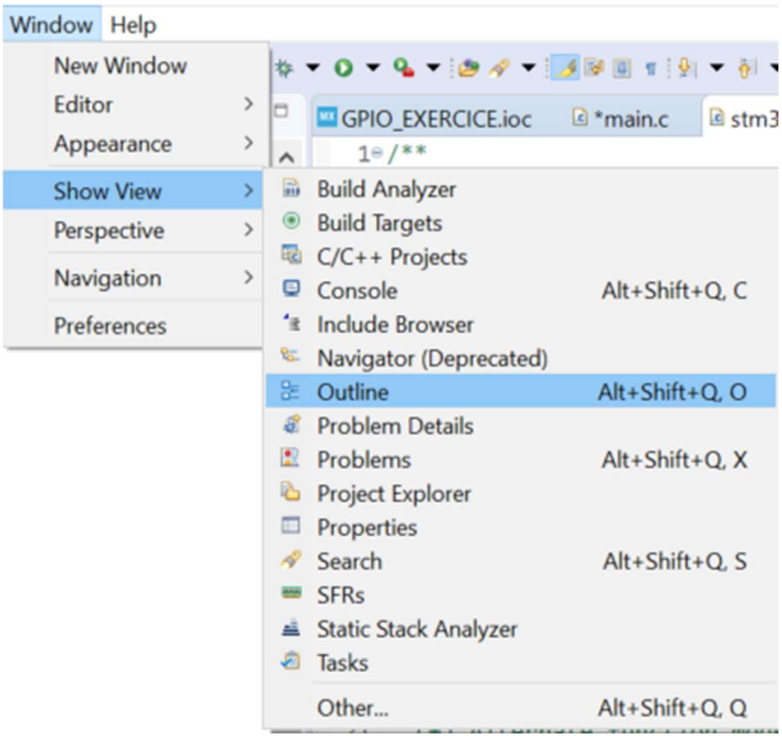
Question :

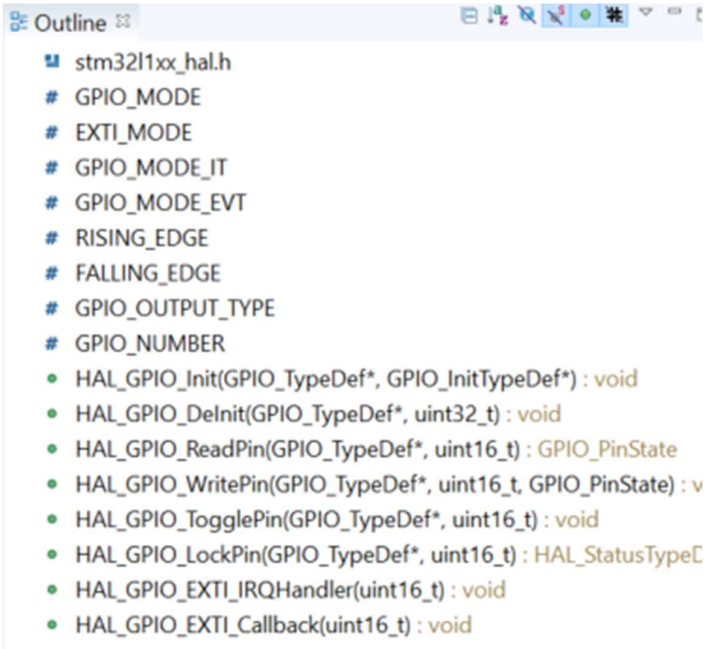
Retrouver la programmation de la fonction Toggle et expliquer son fonctionnement.. Quel registre a été choisi pour changer le niveau de la diode : ODR ou BSRR ?

Les fonctions peuvent être nombreuses. Elles sont classées par périphériques. Les fonctions sont définies dans le registre STM32L1xx_HAL_Driver/Src



Double cliquer sur le fichier stm32l1xx_hal_gpio.c. Puis dans le menu Window, Show View/Outline, la fenêtre outline permet de visualiser toutes les variables et fonctions définies dans un fichier donné.





Toutes ces fonctions sont également regroupées dans un manuel d'utilisation UM1816

A la page 205, on retrouve la définition de la fonction HAL_GPIO_TogglePin

HAL_GPIO_TogglePin

Function name	**void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Toggles the specified GPIO pin.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral for STM32L1XX family devices • **GPIO_Pin:** specifies the pins to be toggled.
Return values	• **None:**

Question :

Quand vous lancez le programme, la LED est éteinte. En trouvant la fonction HAL correcte, forcer la broche PA5 à haut pour allumer la LED avant de rentrer dans la boucle infinie.

Lors de la programmation en assembleur, nous devions programmer si une broche était en sortie ou en entrée, Si elle était en sortie, nous devions la configurer en push-pull ou Open Drain. Ici, tout a été configuré automatiquement. Lors de la création du projet, une section intitulée « initialize all configured peripherals », la fonction MX_GPIO_Init() est appelée

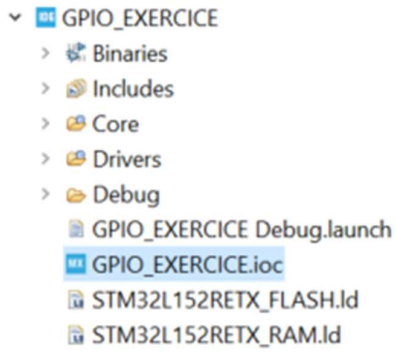
```
/* Initialize all configured peripherals */
MX_GPIO_Init();
```

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}
```

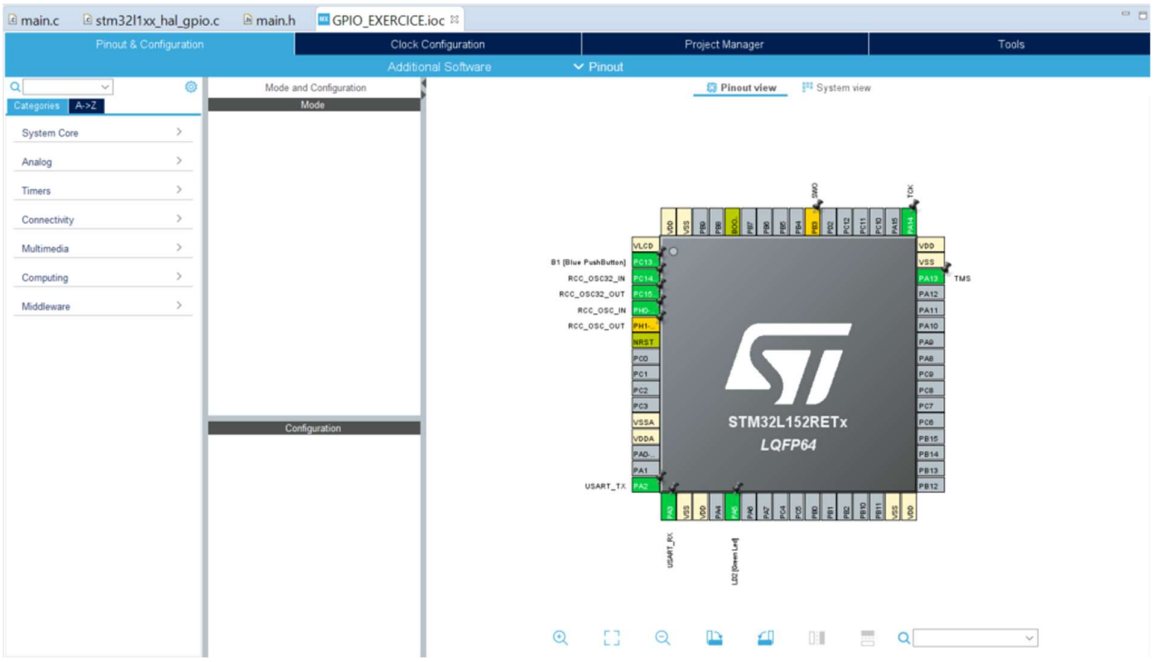
Question :
Sans regarder l'implémentations des fonctions et macros, comprendre ce que fait la fonction MX_GPIO_Init.
Comment est implémenter LD2_Pin ?

Vous devez découvrir que des noms de fonction bien choisis et des commentaires permettent de comprendre un programme, même sans connaître l'implémentation de ce programme.

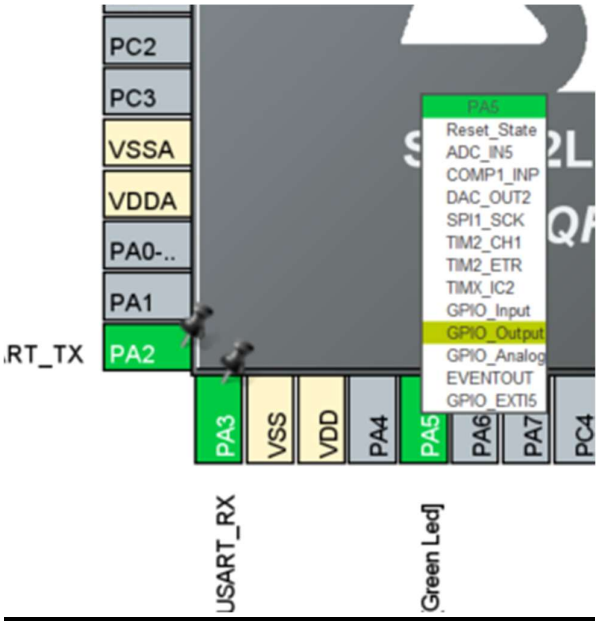
Toute cette configuration a été générée grâce à l'interface graphique CubeMX.
Dans votre projet, cliquer sur le 'nomDeVotreProjet'.ioc file.



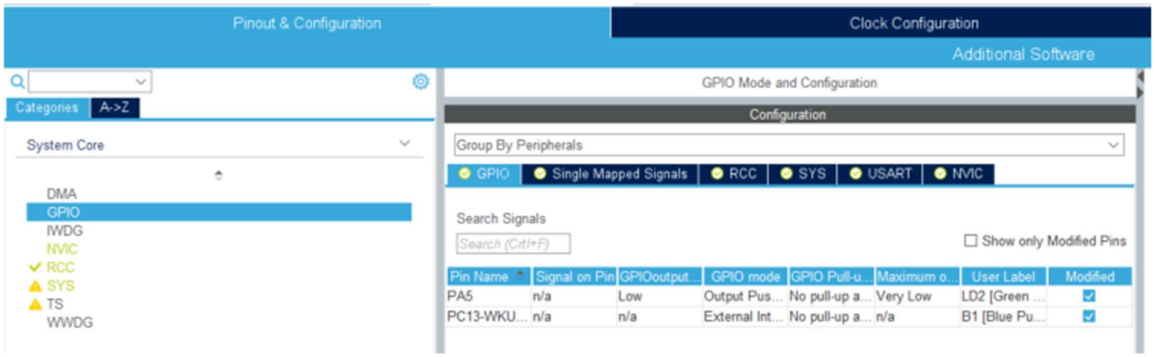
Une fenêtre appelée CubeMX doit s'ouvrir.



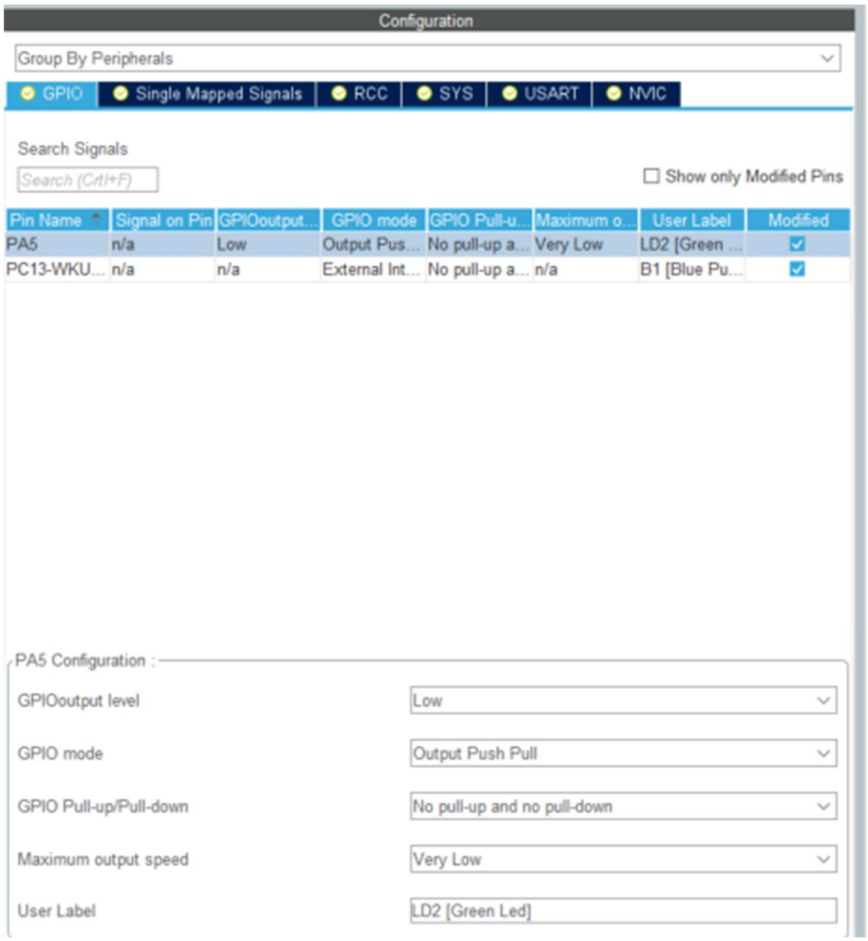
Un click sur chaque broche permet de voir toutes les fonctionnalités disponibles pour une broche donnée. Nous voyons que la broche PA5 est configuré comme GPIO_Output.



Sur la gauche, cliquer sur System Core puis GPIO pour faire apparaître la fenêtre de configuration du GPIO.



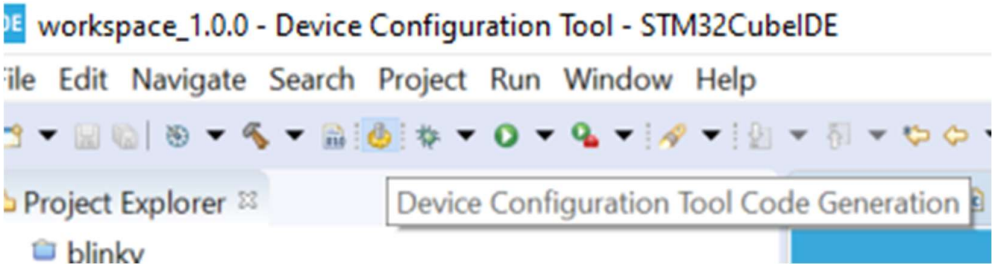
Un nouveau click sur la broche souhaitée (PA5 dans notre cas) permet de visualiser tous les détails de la configuration de la broche.



Question :

Configurer la broche ne mode Haut pour que la LED soit allumée lors de l'initialisation.

Pour générer de nouveau le code, cliquer sur l'icône 'Device Configuration Tool Code Generation'.



Compiler le programme et lancer le debuggeur.

Bien comprendre tout ce processus. Nous utiliserons le cubeMX et les fonctions HAL dans les futurs projets.

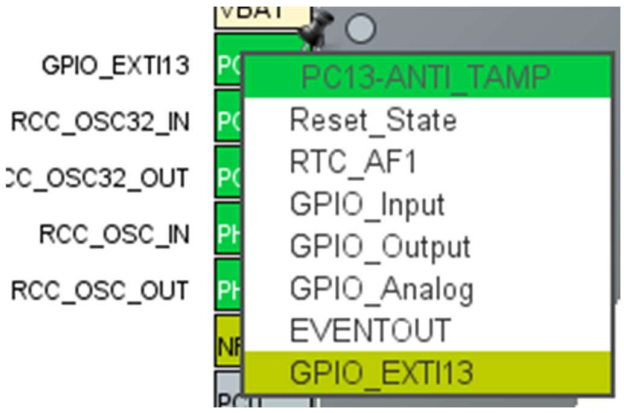
Chapitre 4 Interruption avec un GPIO

Sur un microcontrôleur, les registres sont reliés à des interruptions. Lors d'une interruption, un programme dédié est appelé. Le nom de ces routines est imposé par ARM et il n'est pas possible de les changer.

Lors d'une interruption par un GPIO, plusieurs blocks doivent être configurés :

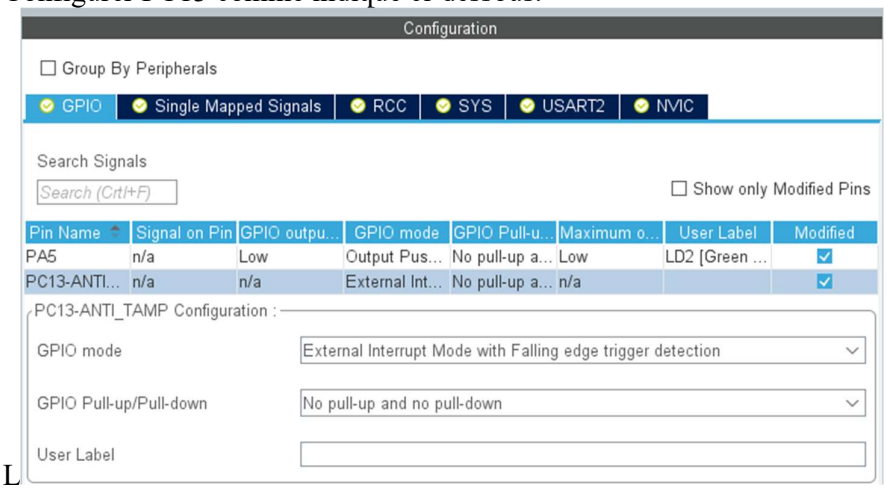
- ✓ Port en entrée et interruption
- ✓ EXTI qui configure la polarité
- ✓ NVIC qui active les interruptions

Le bouton bleu est sur la broche PC13.
Créer une nouveau projet avec le cubeMX et configurer la proche PC13 comme GPIO_EXTI3



Confirmer avec la documentation à quel EXTI la broche 13 est connectée.

Configurer PC13 comme indiqué ci-dessous.



L

Question :

Trouver la nom de l'interruption qui est appelée quand une interruption se produit sur la broche PC13 en appuyant sur la bouton bleu. Le code dans l'interruption permet de faire basculer la valeur de LED verte.
STMicroelectronics fournit un exemple d'utilisation de EXTI lab. Se référer à cet exemple (attention, le composant, les broches, les numéros d'interruption… sont différents).
Télécharger le fichier exemple_HAL_GPIO_EXTI.pdf sur 'ENT.

Ecrire le programme dans la section User Code Begin 4

```
/* USER CODE BEGIN 4 */


/* USER CODE END 4 */
```

Mettre un point d'arrêt dans le code de l'ISR. Noter les valeurs des registres ISER dans le NVIC et les registres du périphérique EXTI.

L'ISER appartient au cœur du micrcontrôleur et est conçu par ARM et non par les vendeurs de silicium. Ce registre est expliqué dans le Programming Manual et non dans le Reference Manual.

Dans le registre xPSR,, le numéro d'interruption est enregistré. Donner cette valeur d'interruption et la comparer avec la documentation de ST. (Indication, comprendre la table de vecteurs d'interruptions dans le start-up file et la table 51 Vector Table du Programming Manual).
Rapprocher cette valeur à la configuration du registre ISER dans dans le NVIC.

Pour activer ou désactiver les interruptions, ARM fournit des fonctions appelées CMIS functions.

Table 21. CMSIS intrinsic functions to generate some Cortex-M4 instructions

Instruction	CMSIS intrinsic function
CPSIE I	void __enable_irq(void)
CPSID I	void __disable_irq(void)
CPSIE F	void __enable_fault_irq(void)
CPSID F	void __disable_fault_irq(void)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
REV	uint32_t __REV(uint32_t int value)
REV16	uint32_t __REV16(uint32_t int value)
REVSH	uint32_t __REVSH(uint32_t int value)
RBIT	uint32_t __RBIT(uint32_t int value)
SEV	void __SEV(void)
WFE	void __WFE(void)
WFI	void __WFI(void)

Faire un programme qui désactive les interruptions après avoir appuyé sur le bouton bleu.

Question :

Quelle est la valeur du registre PRIMASK ?

Annexe Créer un projet STM32 avec CubeMX

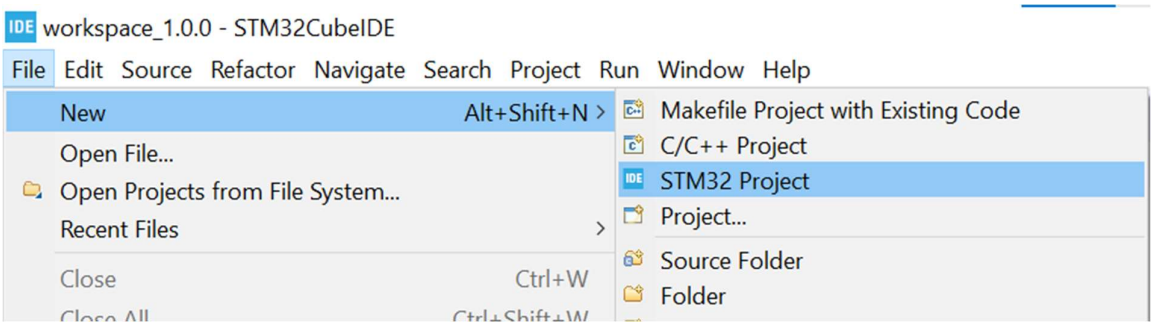
Le kit de développement de ST Microelectronics dédié aux microcontrôleurs des familles STM32 permet de créer très simplement des projets avec un environnement graphique.

Ce livret vous guidera pas à pas dans la création d'un tel projet.

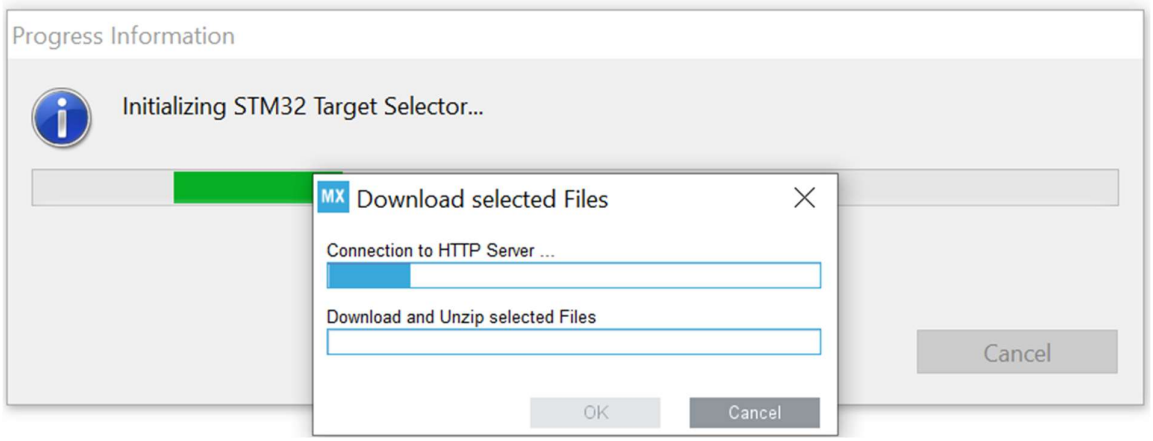
Chaque étape sera détaillée et illustrée de captures d'écran afin de faciliter votre lecture.

Après avoir lu ce petit guide, vous serez capable de créer votre premier projet et de le tester

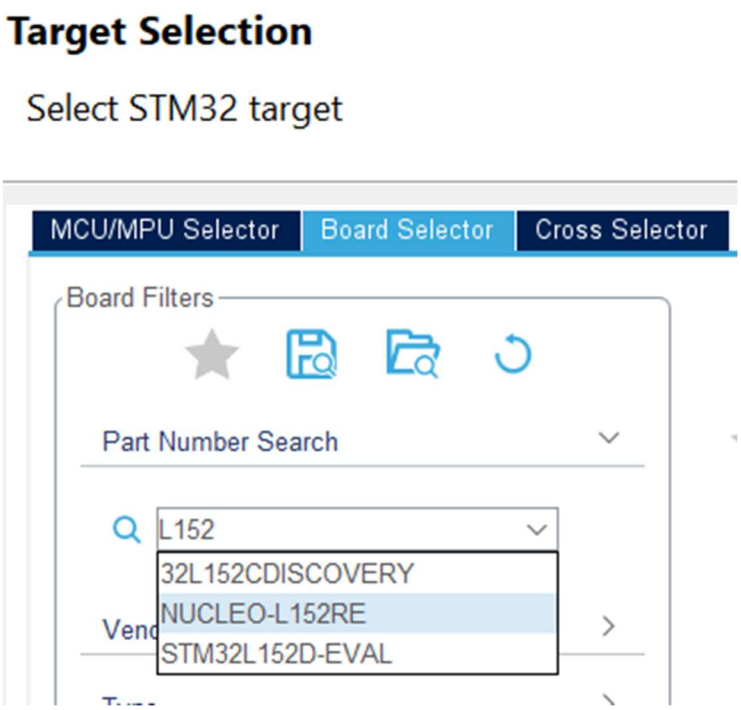
Pour créer un nouveau projet, cliquer sur File/New/STM32 Project



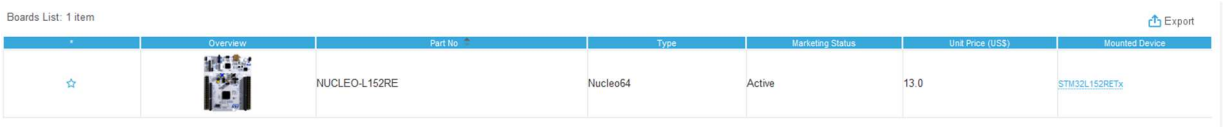
Si une connection à internet est disponible, STM32CubeIDE peut mettre à jour l'offre des produits disponible. Cette mise à jour peut prendre quelques secondes.



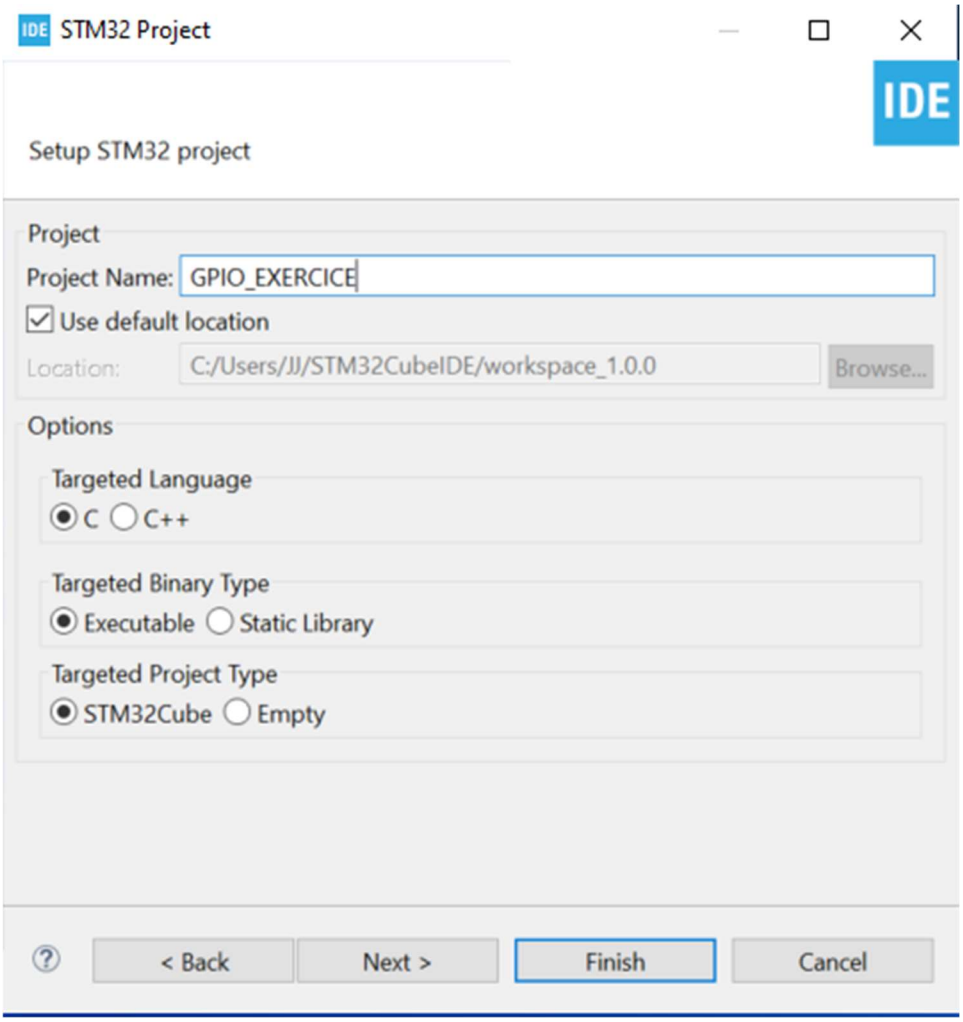
Sélectionner la tabulation Board Selector et dans la fenêtre de recherche Parr Number Research, sélectionner Nucleo-L152RE



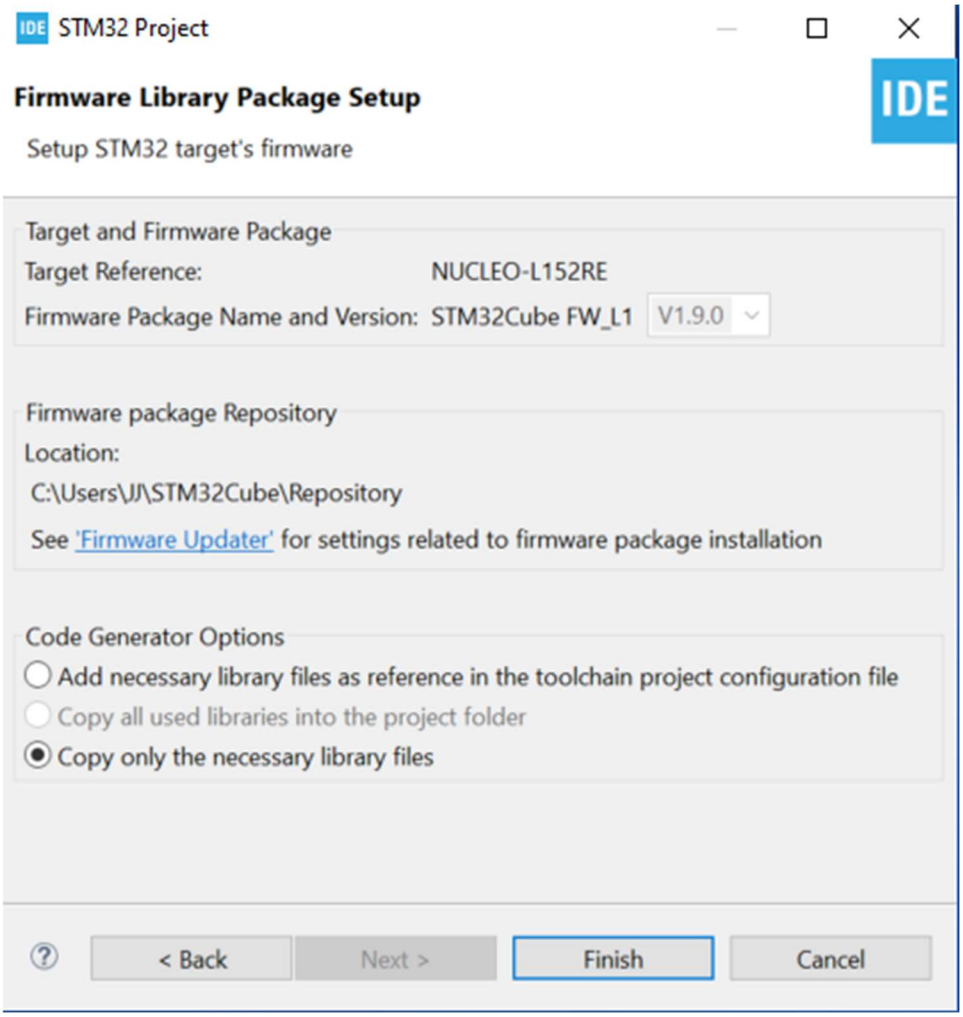
A droite de la fenêtre, la carte utilisée pour le TP apparaît.



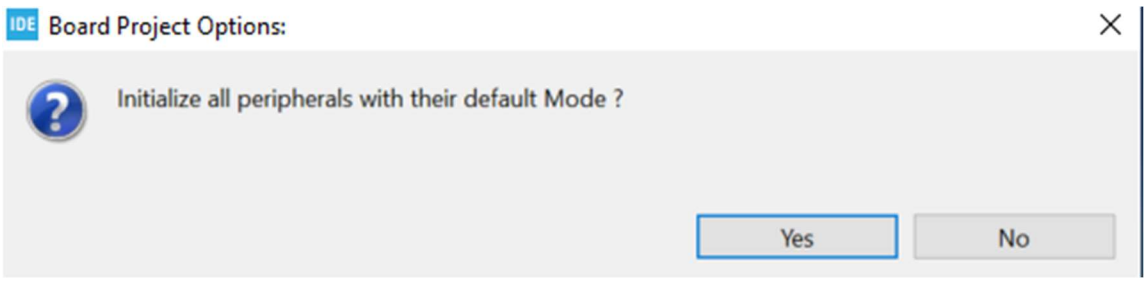
Entrer le nom du projet.



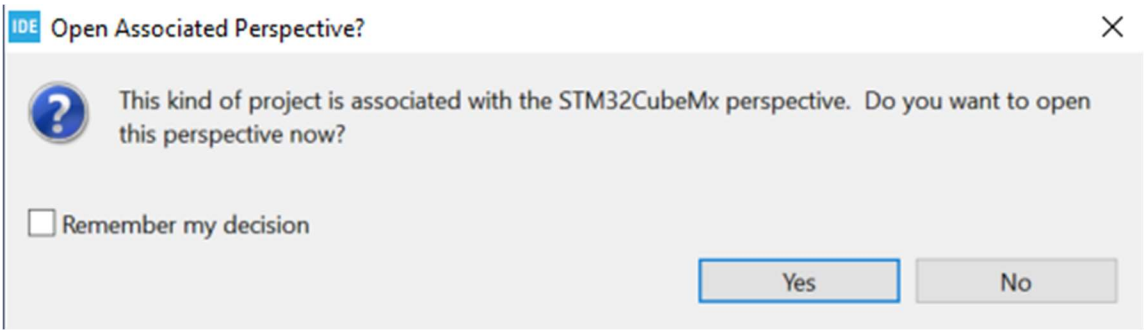
En cliquant sur Next, on peut vérifier que la librarie STM32Cube FW_L1 est bien installée



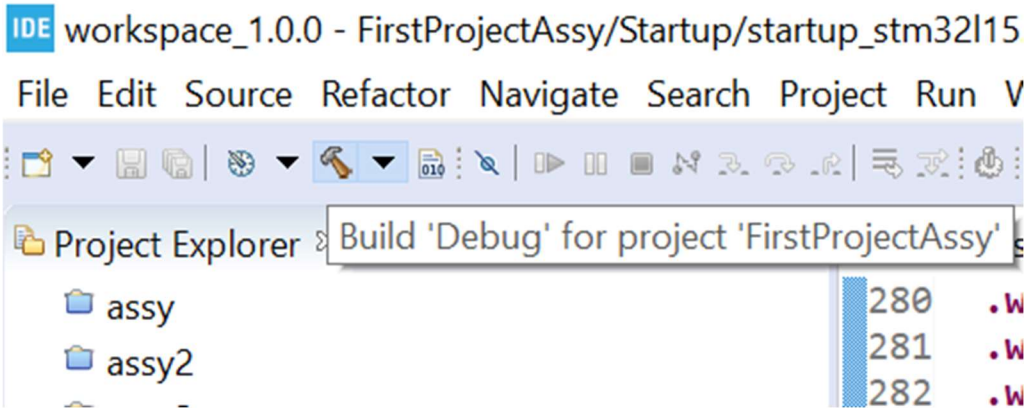
Cliquer sur Finish
Initialiser tous les périphérique à leur mode par défaut



Ouvrir la perspective STM32CubeMx.



Cette perspective permet de configurer les périphériques grâce à une interface graphique et de générer le code cette configuration.
Puis compiler votre programme




```
arm-none-eabi-size   FirstProjectAssy.elf
arm-none-eabi-objdump -h -S  FirstProjectAssy.elf  > "FirstProjectAssy.list"
   text    data     bss     dec     hex filename
    536       8    1568    2112     840 FirstProjectAssy.elf
arm-none-eabi-objcopy  -O binary  FirstProjectAssy.elf  "FirstProjectAssy.bin"
Finished building: default.size.stdout

Finished building: FirstProjectAssy.list

Finished building: FirstProjectAssy.bin


10:29:05 Build Finished. 0 errors, 0 warnings. (took 1s.491ms)
```

Sans erreur, vous êtes prêt à déboguer votre programme.