Université IBN TOFAIL – Kénitra
ECOLE NATIONALE DES SCIENCES APPLIQUEES

Cours Microcontrôleurs Avancés

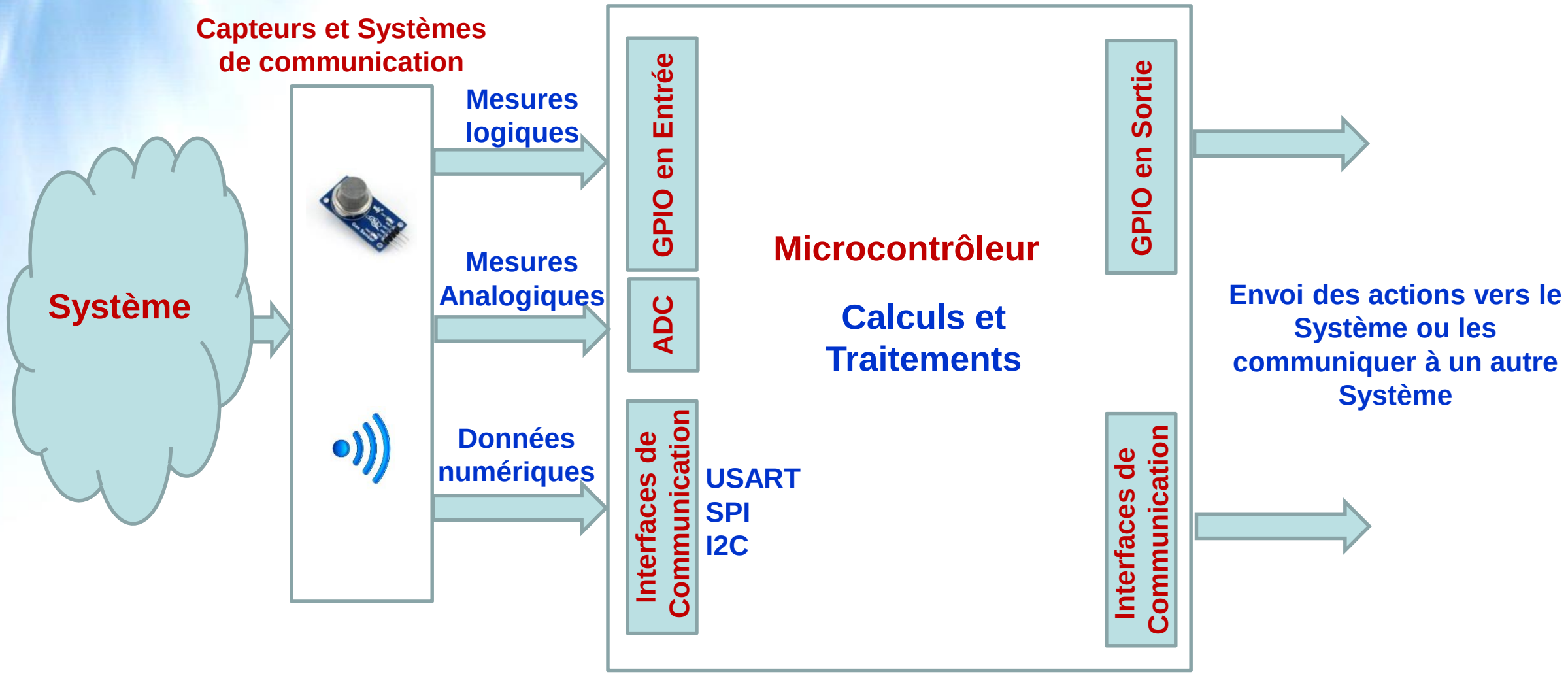
CH3: Projet STM32 CubeIDE: LE GPIO en Sortie

Par:

Hassan EL FADIL

elfadil.h@gmail.com

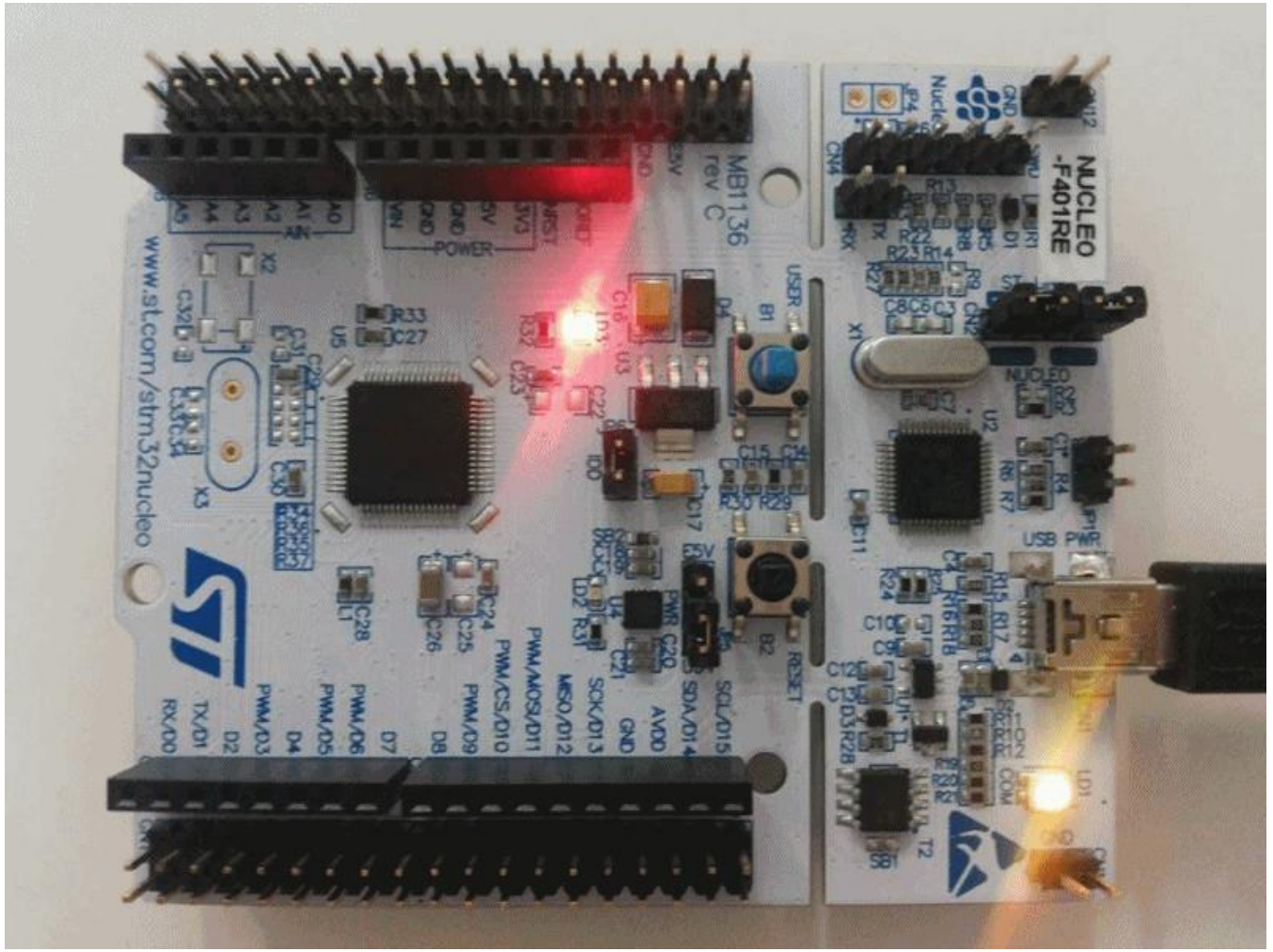
Architecture d'une commande par microcontrôleur



Première Application : Sortie GPIO

Cahier des Charges:

- Programmer le clignotement de la LED utilisateur LED Verte de la carte
- La fréquence de clignotement est réglable par l'utilisateur



GPIO : General-purpose input/outputs

Chacun des GPIO dispose de **quatre** registres **32 bits** de contrôle (GPIOx_MODER,GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR) pour configurer jusqu'à 16 E/S.

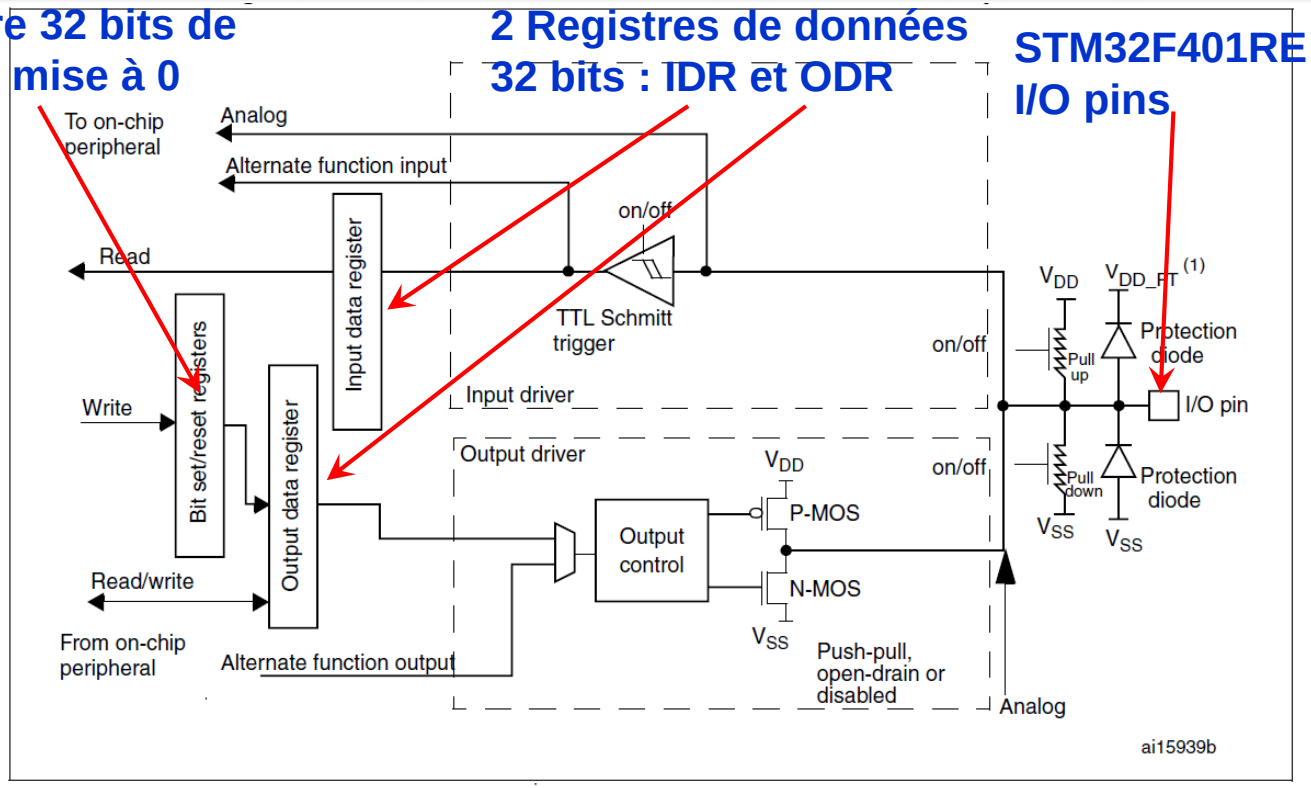
- Le registre **GPIOx_MODER** permet de sélectionner le sens des E/S (entrée, sortie, AF: alternate func, analogique).
- Les registres **GPIOx_OTYPER et GPIOx_OSPEEDR** permettent de sélectionner le type de sortie(push-pull ou open-drain) et la vitesse (les broches de vitesse d'E/S sont directement connectées aubits de registre GPIOx_OSPEEDR correspondants quel que soit le sens des E/S).
- Le registre **GPIOx_PUPDR** permet de sélectionner le pull-up/pull-down quel que soit le sens des E/S.

(x = A,B, C, D and H)

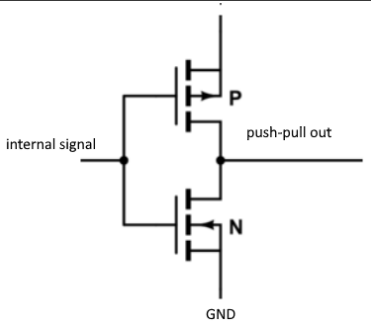
un registre 32 bits de mise à 1 / mise à 0

2 Registres de données 32 bits : IDR et ODR

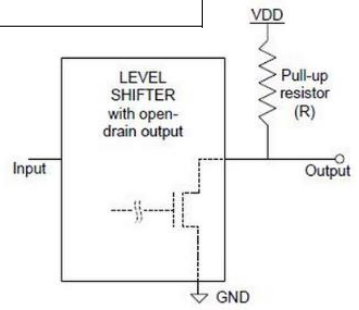
STM32F401RE: 51 GPIO I/O pins



Push-Pull

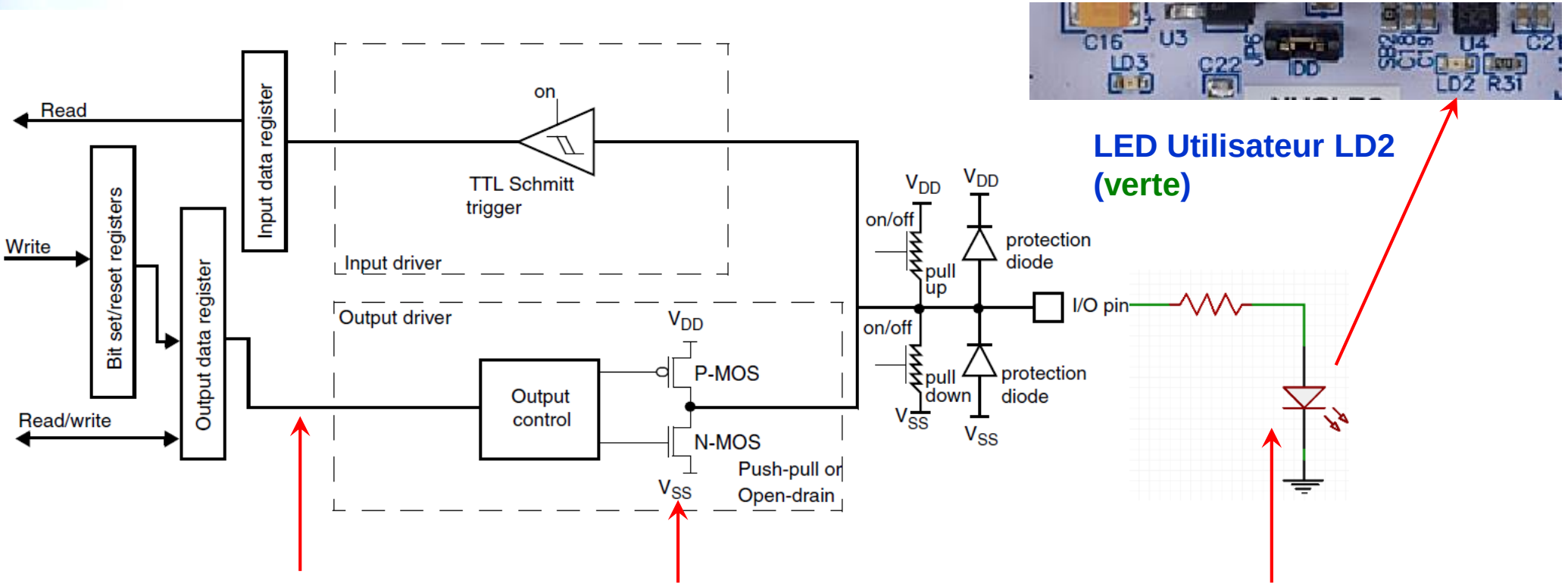


Open Drain (Drain Ouvert)



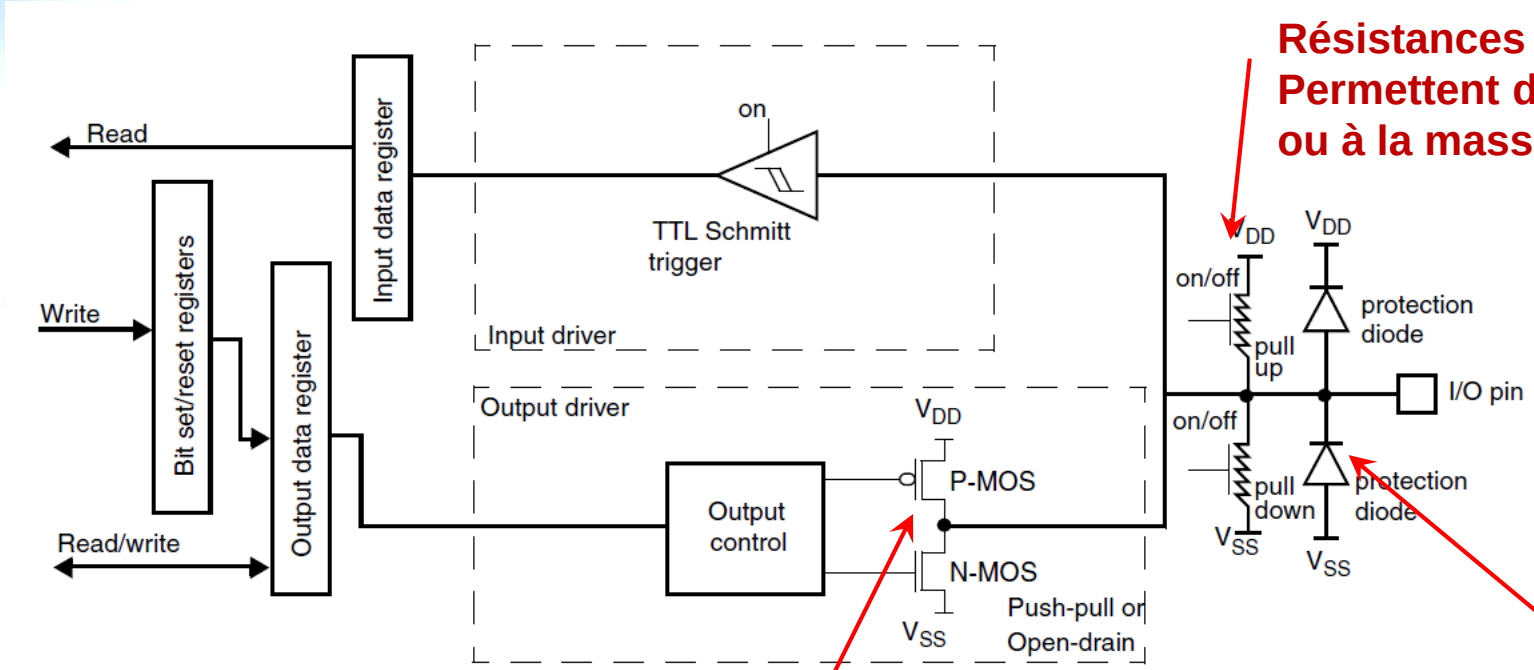
Le GPIO

Configuration Push-Pull par exemple:



Le GPIO

En Configuration Open-Drain le P-MOS est dans l'état Haute impédance:



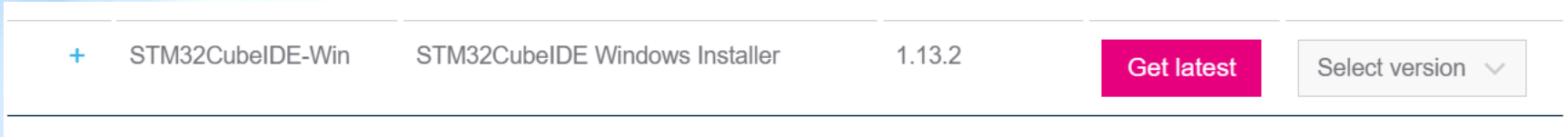
Résistances de Pull-up: Permettent de forcer la pin à VDD ou à la masse

Le P-MOS est à l'état Haute impédance en mode Open-Drain

Diodes de protection: Evitent que la tension de la pin n'excède pas Vdd ni inférieure à la masse.

Logiciel: STM32CubeIDE

- **STM32CubeIDE**, le IDE libre pour developer les programmes STM32
- Lien pour télécharger: https://www.st.com/en/development-tools/stm32cubeide.html



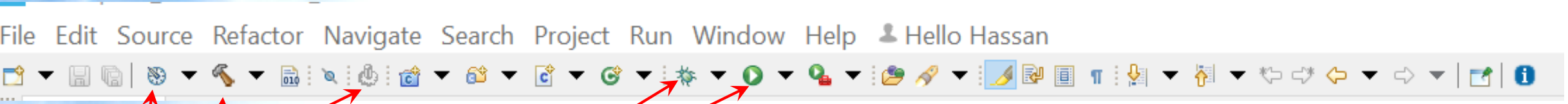
Vous devez vous inscrire à STM pour télécharger le logiciel, les mises à jour et l'installation de certains freeware.

IDE (Integrated Development Environment) signifie Environnement de Développement Intégré.

Voici quleques fonctionnalités de l'IDE:

- Éditeur de code
- Compilateur/Interprète : traduit le code source écrit par le programmeur en code machine ou bytecode
- Débogueur : permet de rechercher et de corriger les erreurs dans le code
- Gestion de projet : l'IDE aide à gérer les projets en organisant les fichiers, les ressources et les dépendances
- Mise en évidence des erreurs
- …

Menus STM32CubeIDE



“ Device Configuration Tool Code Generation ” : Pour générer le code

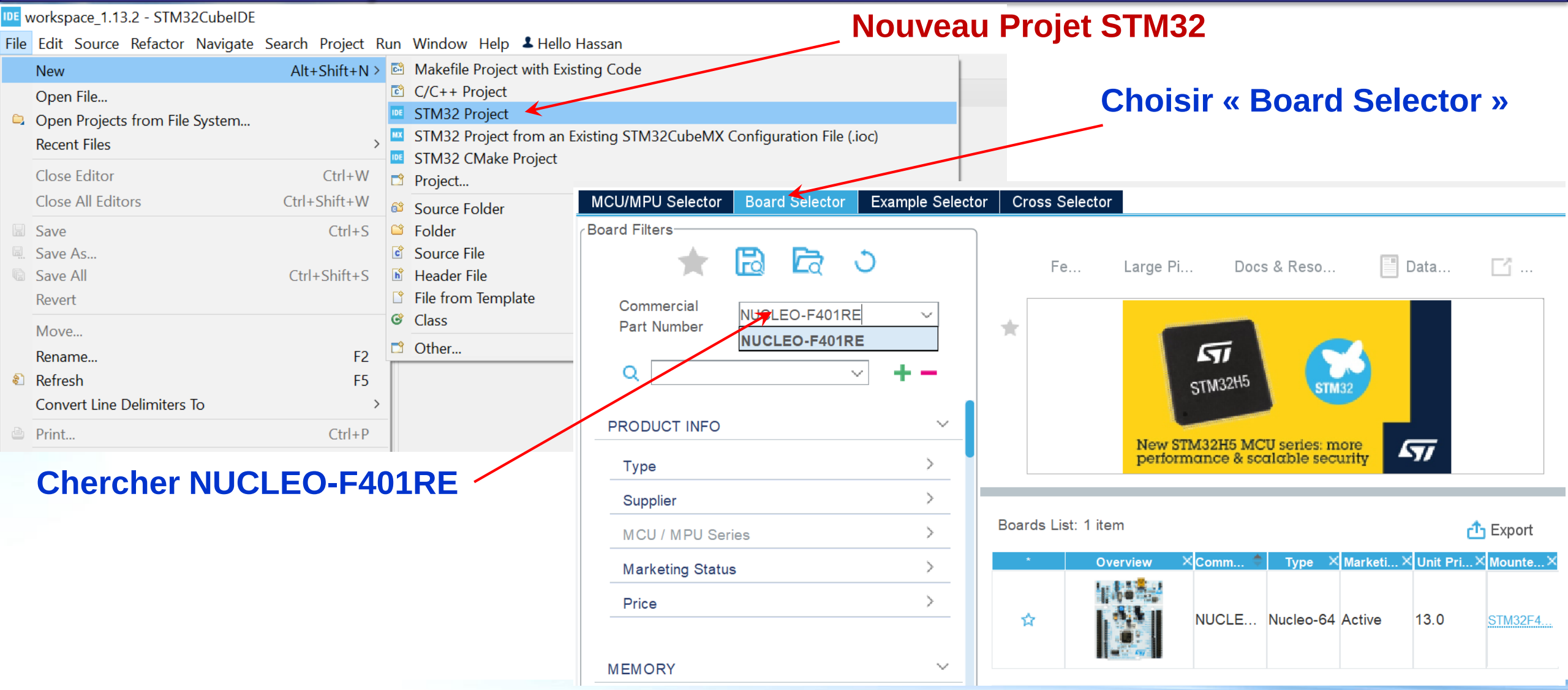
“Run”
Lance l'exécution de votre application sur le microcontrôleur STM32 connecté. Lorsque vous sélectionnez cette option, STM32CubeIDE compile votre code (si nécessaire), charge le binaire sur le microcontrôleur et commence à exécuter votre application. Cette option est utilisée lorsque vous souhaitez démarrer l'application et la laisser s'exécuter sans pause pour le débogage.

“Debug”
Ces options permettent de contrôler le processus de débogage, permettant aux développeurs d'analyser de manière interactive le comportement de leur code, d'identifier les problèmes et de vérifier l'exactitude de leurs applications sur les microcontrôleurs STM32. (START, STOP, RESUME, PAUSE, RESTART, BREAKPOINT, …)

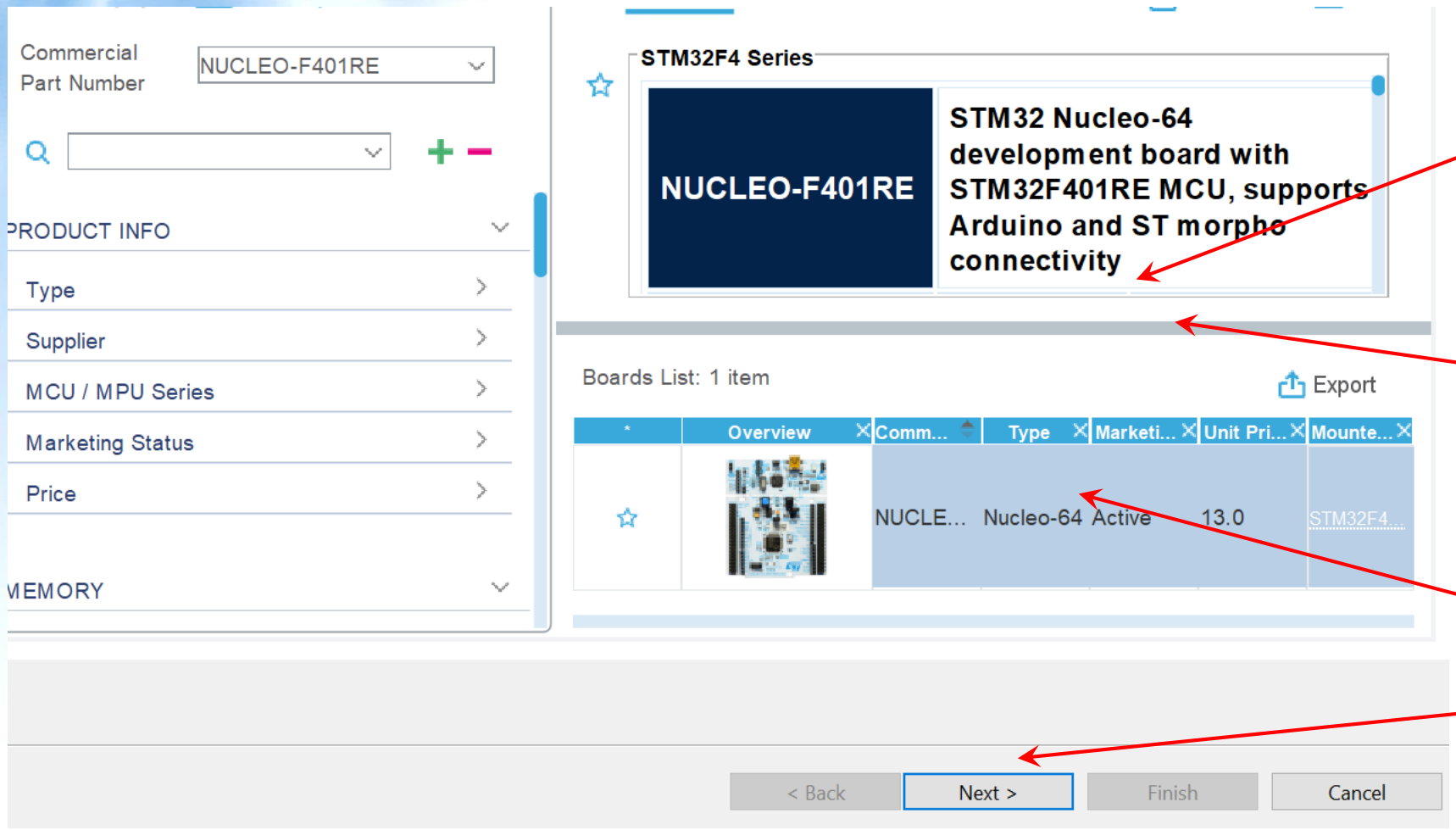
“Build Debug”
Compilation les fichiers de code source sont compilés en fichiers objets, Linking : les fichiers objets sont liés entre eux pour créer un fichier exécutable.

“Manage Configurations”
Cette fonctionnalité permet de créer, modifier et supprimer différentes configurations de build.

Nouveau Projet STM32CubeIDE



Nouveau Projet STM32CubeIDE

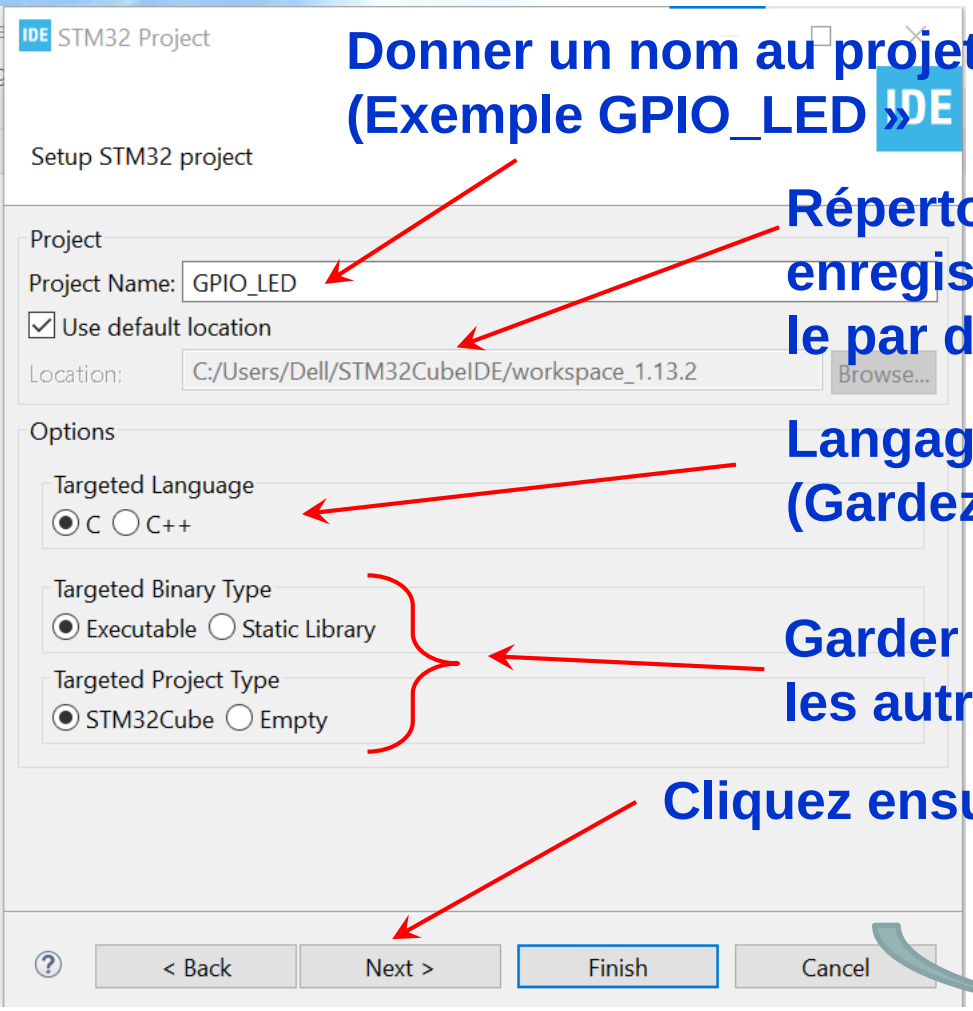


Vous pouvez visualiser ici les informations sur la carte

Vous pouvez allonger cette partie avec la sourie pour voir plus de détails

Sélectionner la carte et cliquer sur « Next »

Nouveau Projet STM32CubeIDE



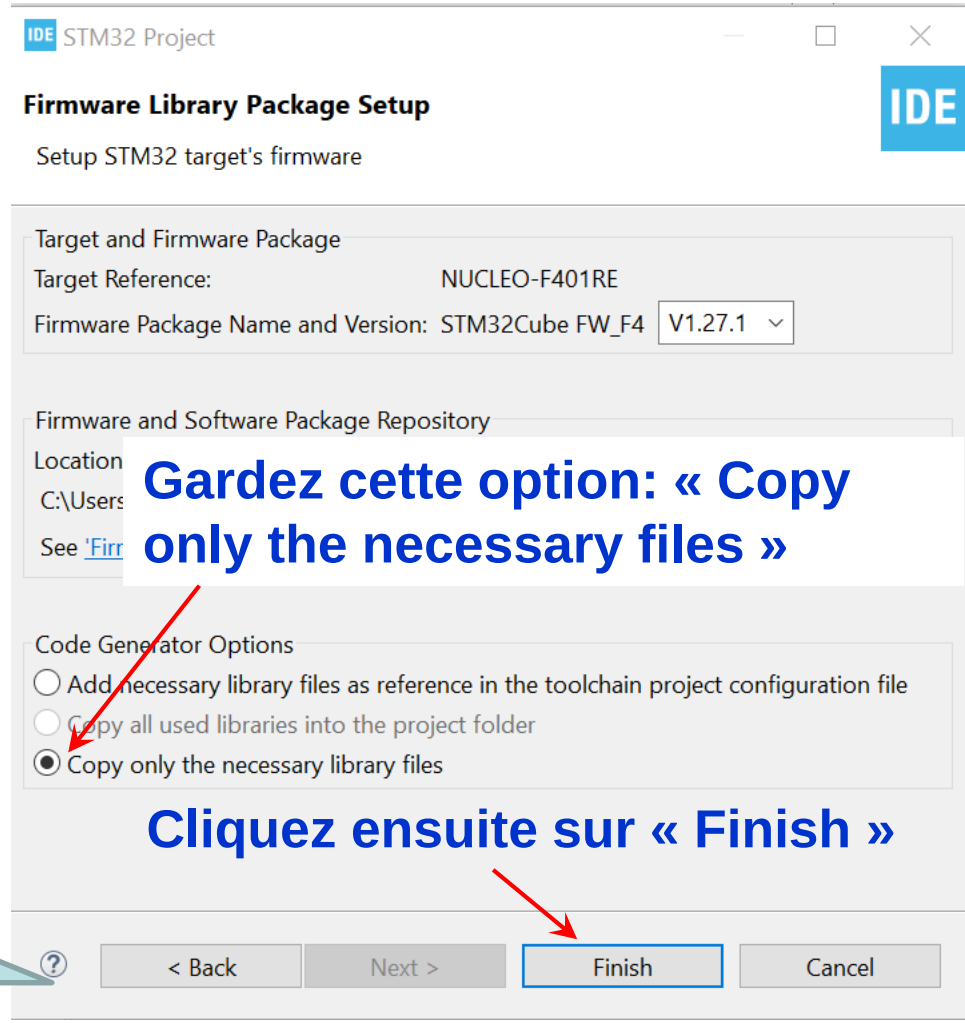
Donner un nom au projet (Exemple GPIO_LED »

Répertoire de travail où sera enregistré votre projet (Gardez le par défaut)

Langage de programmation (Gardez le par défaut: C)

Garder par défaut les autres parties

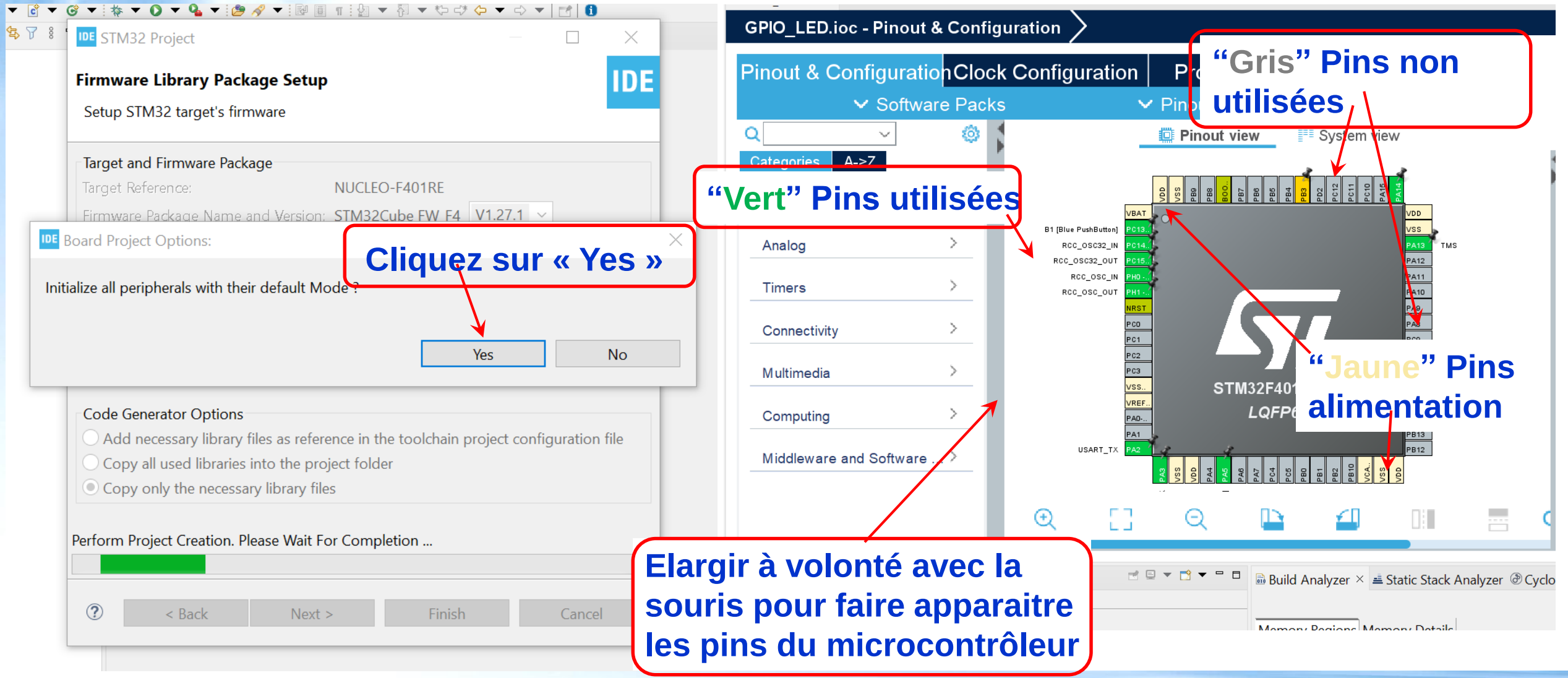
Cliquez ensuite sur « Next »



Gardez cette option: « Copy only the necessary files »

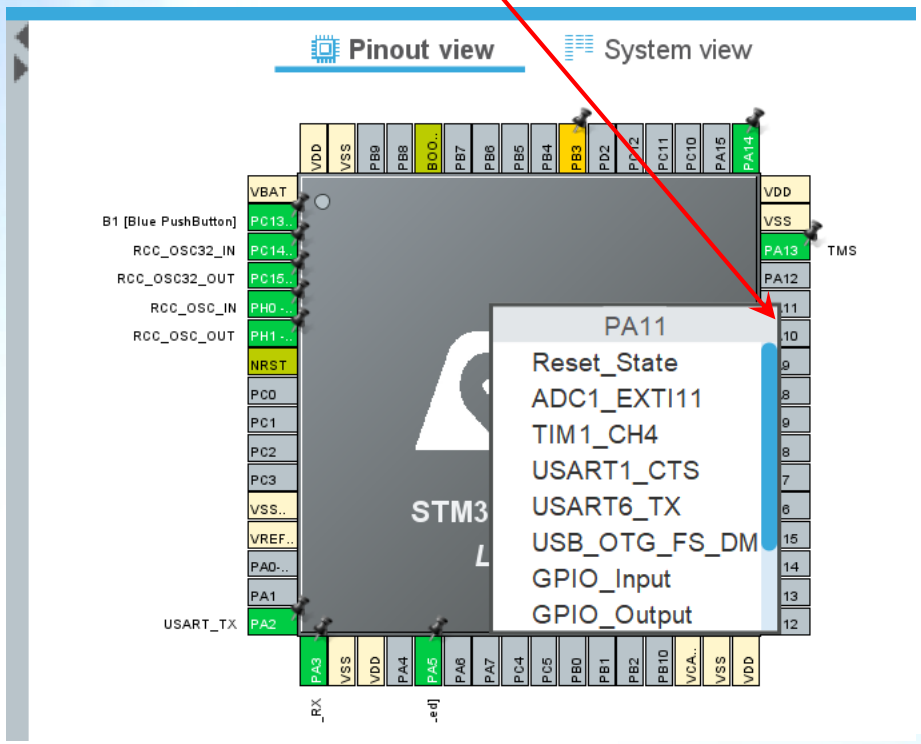
Cliquez ensuite sur « Finish »

Nouveau Projet STM32CubeIDE

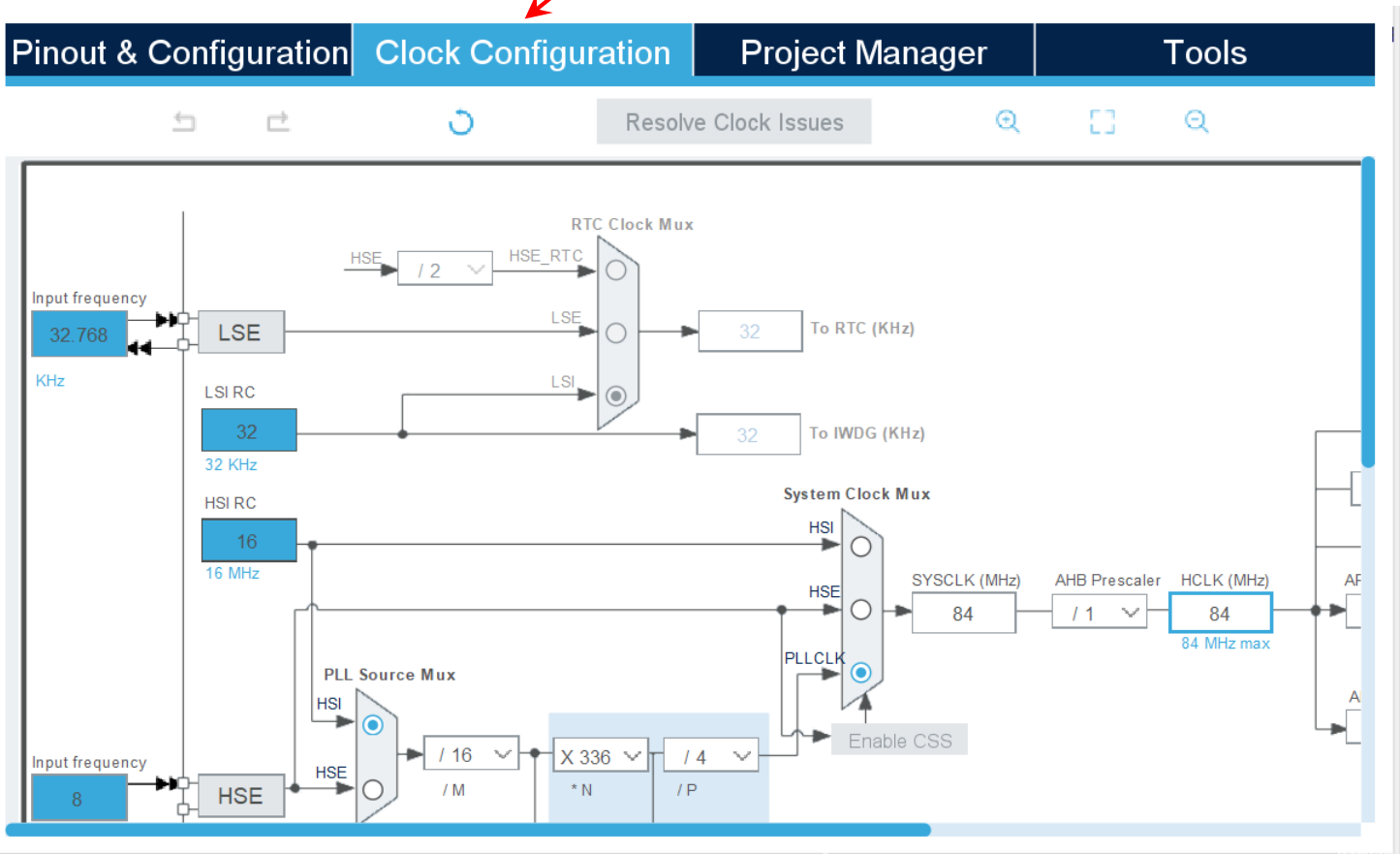


Nouveau Projet STM32CubeIDE

Vous pouvez cliquer sur une pin pour la configurer: en entrée, en sortie ou autre tâche.
Pour le moment laissez les choses par défaut.

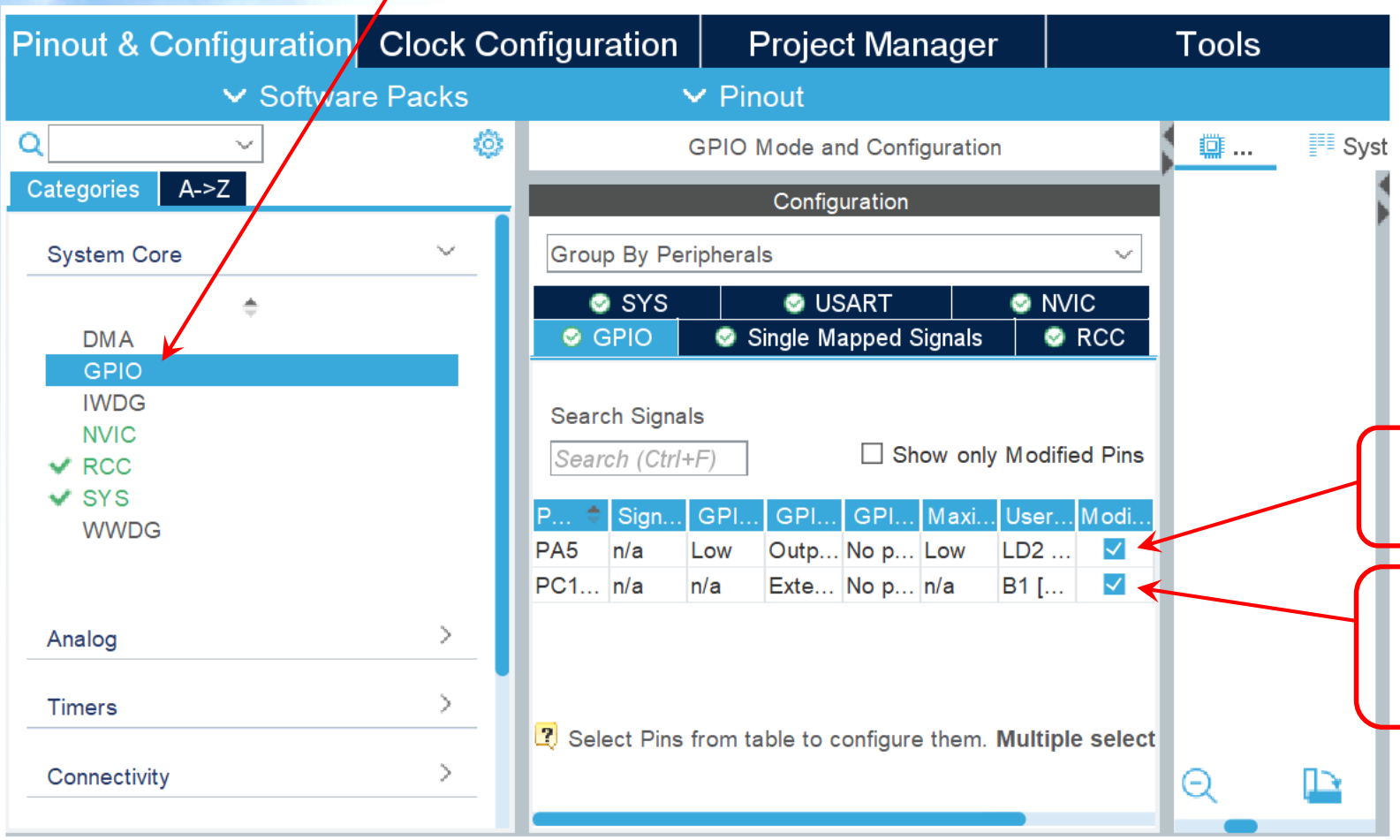


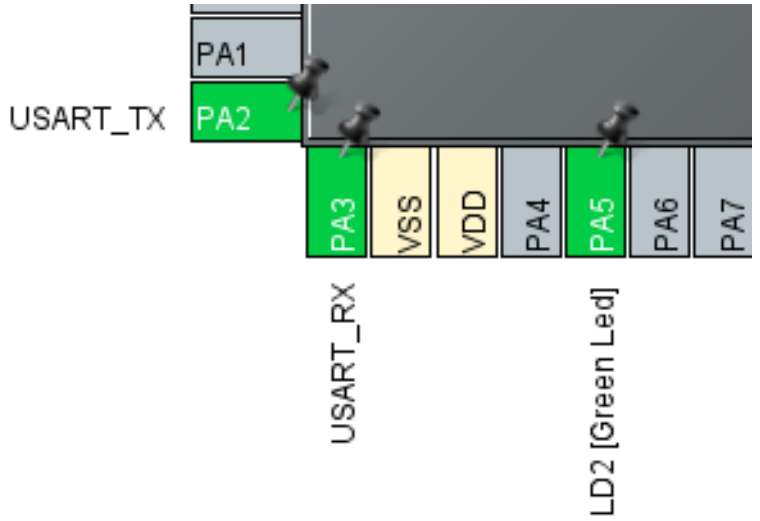
Laissez « par défaut » la configuration des horloges.
Nous allons voir leurs utilités par la suite



Configuration du GPIO

Sélectionner « System Core », GPIO.

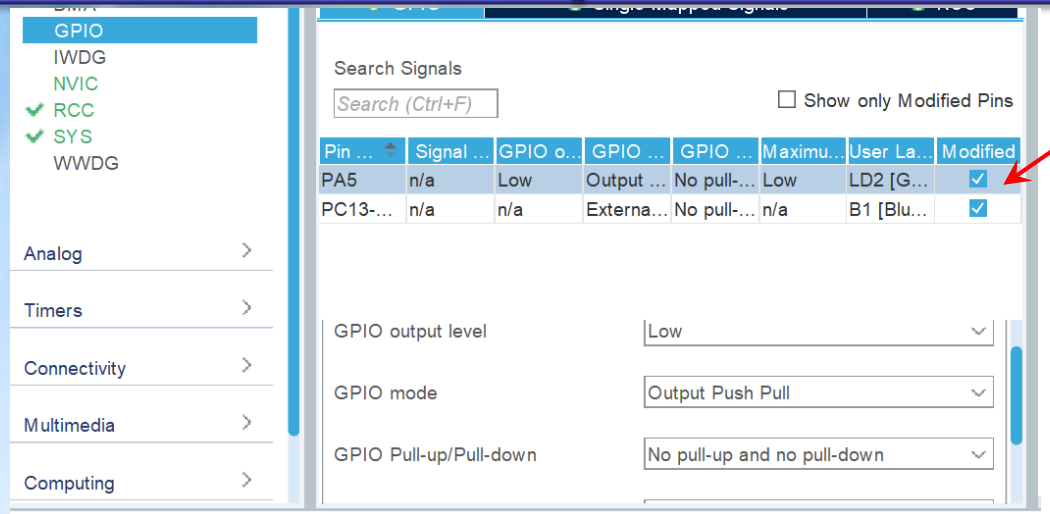




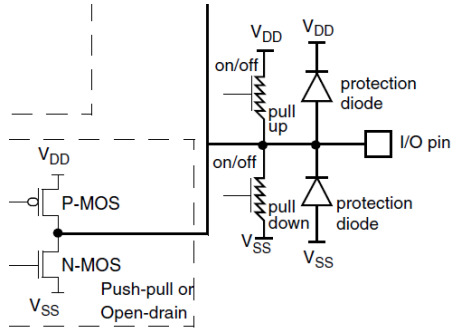
Pin PA5 déjà configurée pour la LED2.

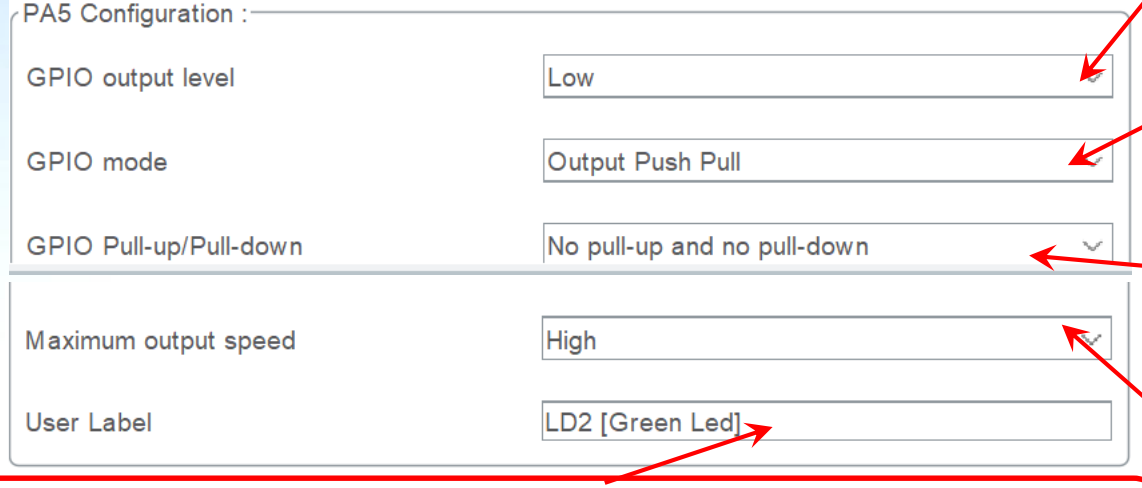
Pin PC1 déjà configurée pour le Bouton Poussoir utilisateur B1 (Bleu).

Configuration du GPIO



Double-Cliquer sur la ligne PA5





Valeur initiale (Garder Low): LED éteinte au départ

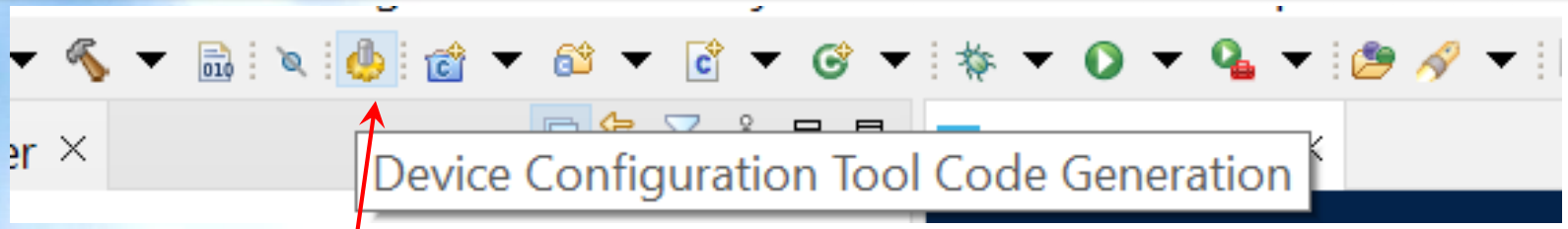
Mode: Push-Pull, Open Drain (Garder Push-Pull)

Activation des résistances de pull-up / pull-down (Garder No)

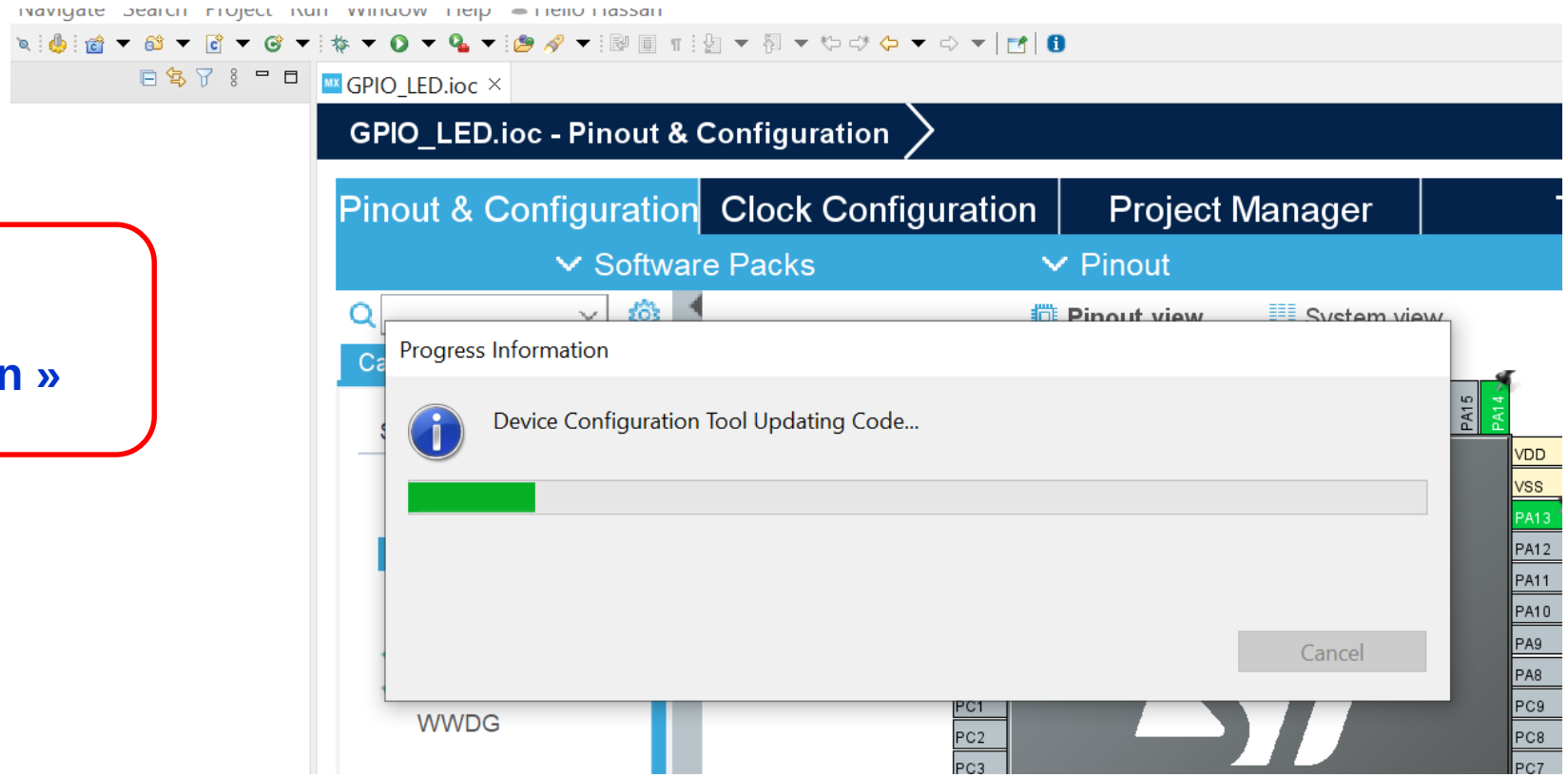
Vitesse du GPIO: Low, Medium, High, Very-High Mettre « High »

Label utilisateur: Vous pouvez lui donner un nom ou le laisser par défaut

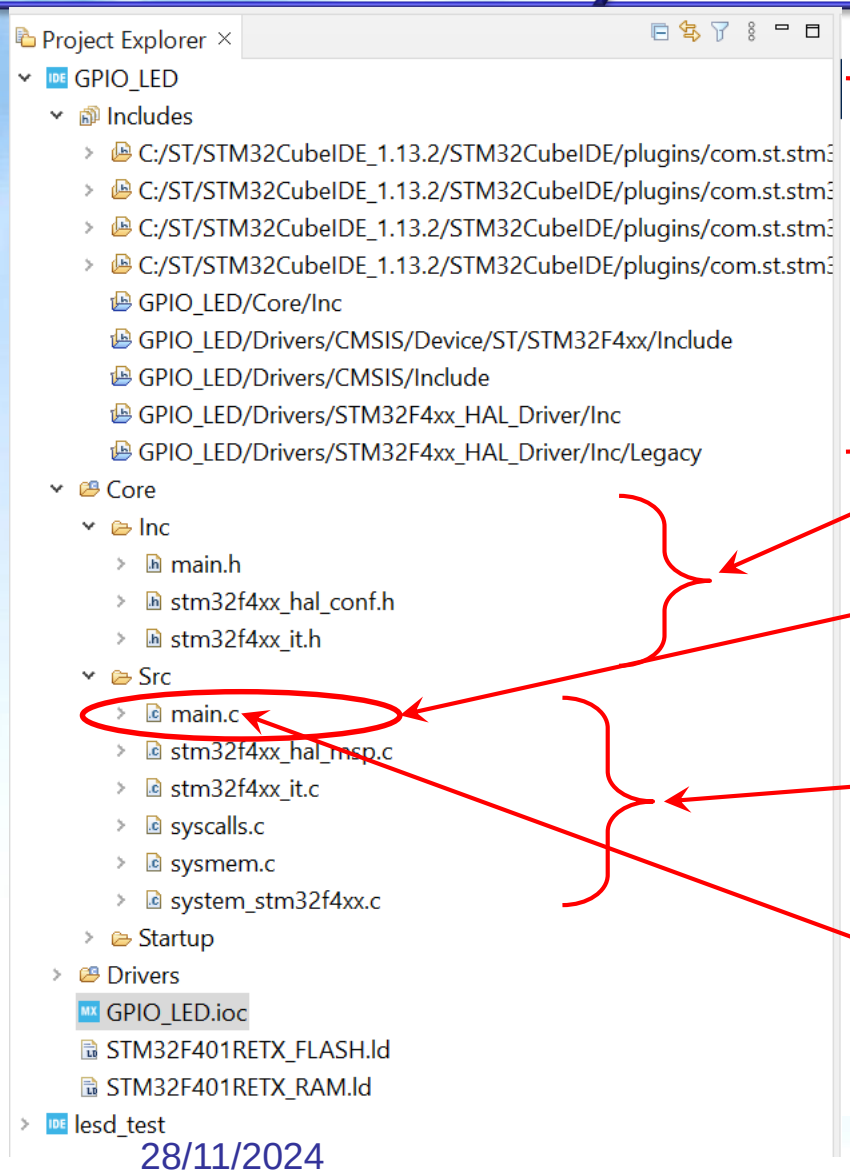
Génération du Code



Cliquer ensuite sur « Device Configuration Tool Code Generation »



Génération du Code



Toutes les Laibrairies incluses dans ce projet

Fichiers avec l'extension .h (Header Files ou Fichiers d'En-tête)

Le programme principal : Fichier Source

Fichiers avec l'extension .c (Source Files ou Fichiers Source) :

- **Les fichiers source contiennent l'implémentation réelle des fonctions, des structures, des variables globales, etc., déclarées dans les fichiers d'en-tête.**
- **Les fichiers source sont compilés par le compilateur C pour générer le fichier exécutable final.**

Double-Cliquer ensuite sur « main.c »

Génération du Code

main.c

```
58 /* USER CODE BEGIN 0 */
59
60 /* USER CODE END 0 */
61
62 /**
63   * @brief  The application entry point.
64   * @retval int
65   */
66 int main(void)
67 {
68   /* USER CODE BEGIN 1 */
69
70   /* USER CODE END 1 */
71
72   /* MCU Configuration--------------------------------------------------------*/
73
74   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
75   HAL_Init();
76
77   /* USER CODE BEGIN Init */
78
79   /* USER CODE END Init */
80
81   /* Configure the system clock */
82   SystemClock_Config();
83
84   /* USER CODE BEGIN SysInit */
85
86   /* USER CODE END SysInit */
87
88   /* Initialize all configured peripherals */
89   MX_GPIO_Init();
90   MX_USART2_UART_Init();
91   /* USER CODE BEGIN 2 */
```

Directives de préprocesseur (Preprocessor Directives):
#include
Déclarations de fonctions (Function Declarations)
void

Fonction main ():

```
int main(void)
{
/* USER CODE BEGIN 1 */
……….;
}
```

Définitions de fonctions :

```
void SystemClock_Config(void)
{
……….;
}
```

Autres fonctions.

Génération du Code (main.c)

Pour voir le contenu d'une Macro: sélectionner la macro et appuyer F3

Activation de toutes les horloges des GPIOs

Fonction permettant de mettre une valeur sur une pin d'un port

GPIO output level — Low

Configuration de de la Pin LD2_Pin

PA5 Configuration :

GPIO output level	Low
GPIO mode	Output Push Pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	High
User Label	LD2 [Green Led]

main.c

```
189   */
190 static void MX_GPIO_Init(void)
191 {
192   GPIO_InitTypeDef GPIO_InitStruct = {0};
193 /* USER CODE BEGIN MX_GPIO_Init_1 */
194 /* USER CODE END MX_GPIO_Init_1 */
195
196   /* GPIO Ports Clock Enable */
197   __HAL_RCC_GPIOC_CLK_ENABLE();
198   __HAL_RCC_GPIOH_CLK_ENABLE();
199   __HAL_RCC_GPIOA_CLK_ENABLE();
200   __HAL_RCC_GPIOB_CLK_ENABLE();
201
202   /*Configure GPIO pin Output Level */
203   HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
204
205   /*Configure GPIO pin : B1_Pin */
206   GPIO_InitStruct.Pin = B1_Pin;
207   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
208   GPIO_InitStruct.Pull = GPIO_NOPULL;
209   HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
210
211   /*Configure GPIO pin : LD2_Pin */
212   GPIO_InitStruct.Pin = LD2_Pin;
213   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
214   GPIO_InitStruct.Pull = GPIO_NOPULL;
215   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
216   HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
217
218 /* USER CODE BEGIN MX_GPIO_Init_2 */
219 /* USER CODE END MX_GPIO_Init_2 */
220 }
221
222 /* USER CODE BEGIN 4 */
223
224 /* USER CODE END 4 */
```

Port, Pin, valeur (ici 0: RESET)

Outli... / Buil...

- main.h
- huart2 : UART_HandleTy
- SystemClock_Config(vo
- MX_GPIO_Init(void) : vo
- MX_USART2_UART_Init(
- main(void) : int
- SystemClock_Config(vo
- MX_USART2_UART_Init(
- MX_GPIO_Init(void) : vo
- Error_Handler(void) : vo
- assert_failed(uint8_t*, ui

Un clique sur un lien, vous ramène vers la fonction

Génération du Code (main.c)

La première fonction a être exécutée dans le programme principal est: main et HAL_Init.

HAL: **H**ardware **A**bstraction **L**ayer (Couche d'abstraction matérielle): Programmes intermédiaire entre le logiciel et le matériel. Il offre des fonctions standardisées de manipulation du matériel tout en cachant les détails techniques de la mise en œuvre.

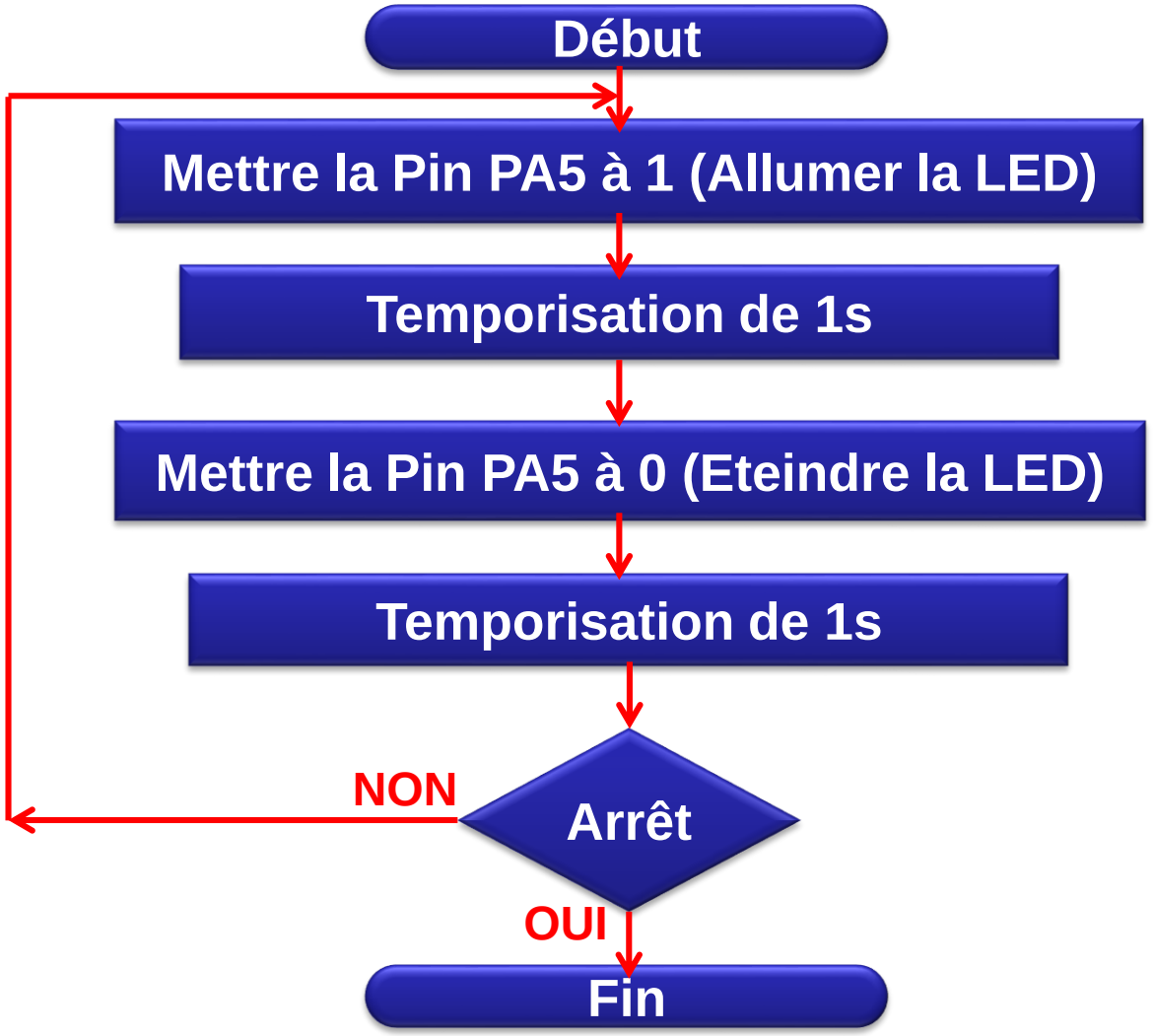
Ici: **HAL_Init** permet une réinitialisation de tous les périphériques, Initialise l'interface Flash et le Systick

Le **SysTick** est un timer intégré. Il s'agit d'un compteur à décompte automatique de 24 bits qui peut être utilisé pour générer des retards, pour mesurer le temps ou pour déclencher des interruptions périodiques.

```
58 /* USER CODE BEGIN 0 */
59
60 /* USER CODE END 0 */
61
62 /**
63   * @brief  The application entry point.
64   * @retval int
65   */
66 int main(void)
67 {
68   /* USER CODE BEGIN 1 */
69
70   /* USER CODE END 1 */
71
72   /* MCU Configuration--------------------------------------------------------*/
73
74   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
75   HAL_Init();
76
77   /* USER CODE BEGIN Init */
78
79   /* USER CODE END Init */
80
81   /* Configure the system clock */
82   SystemClock_Config();
83
84   /* USER CODE BEGIN SysInit */
85
86   /* USER CODE END SysInit */
87
88   /* Initialize all configured peripherals */
89   MX_GPIO_Init();
90   MX_USART2_UART_Init();
91   /* USER CODE BEGIN 2 */
```

Configuration d'horloge

Organigramme du programme



Insertion du code utilisateur dans main.c et compilation

Il faut s'assurer d'écrire votre code dans cet emplacement (entre /* USER CODE BEGIN 3 */ Et /* USER CODE END 3 */) Sinon il sera supprimé lors de la generation du code "Code Generation"

```
 93   /* USER CODE END 2 */
 94
 95   /* Infinite loop */
 96   /* USER CODE BEGIN WHILE */
 97   while (1)
 98   {
 99     /* USER CODE END WHILE */
100
101     /* USER CODE BEGIN 3 */
102   }
103   /* USER CODE END 3 */
104 }
```

```
 97   while (1)
 98   {
 99     /* USER CODE END WHILE */
100
101     /* USER CODE BEGIN 3 */
102       HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); /* Mettre la pin LD2_pin à 1 */
103       HAL_Delay(1000); /* Temporistaion de 1000ms=1s */
104       HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); /* Mettre la pin LD2_pin à 0 */
105       HAL_Delay(1000); /* Temporistaion de 1000ms=1s */
106   }
107   /* USER CODE END 3 */
108 }
```

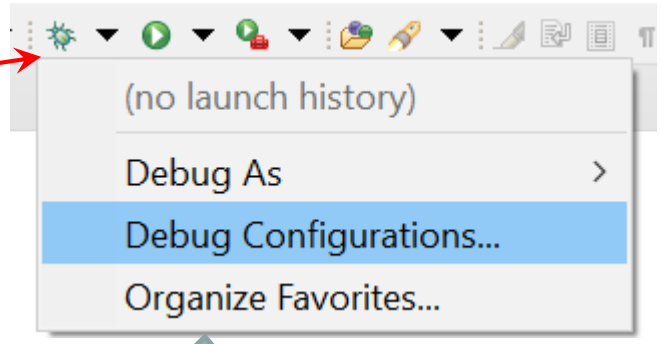
Lancer la compilation

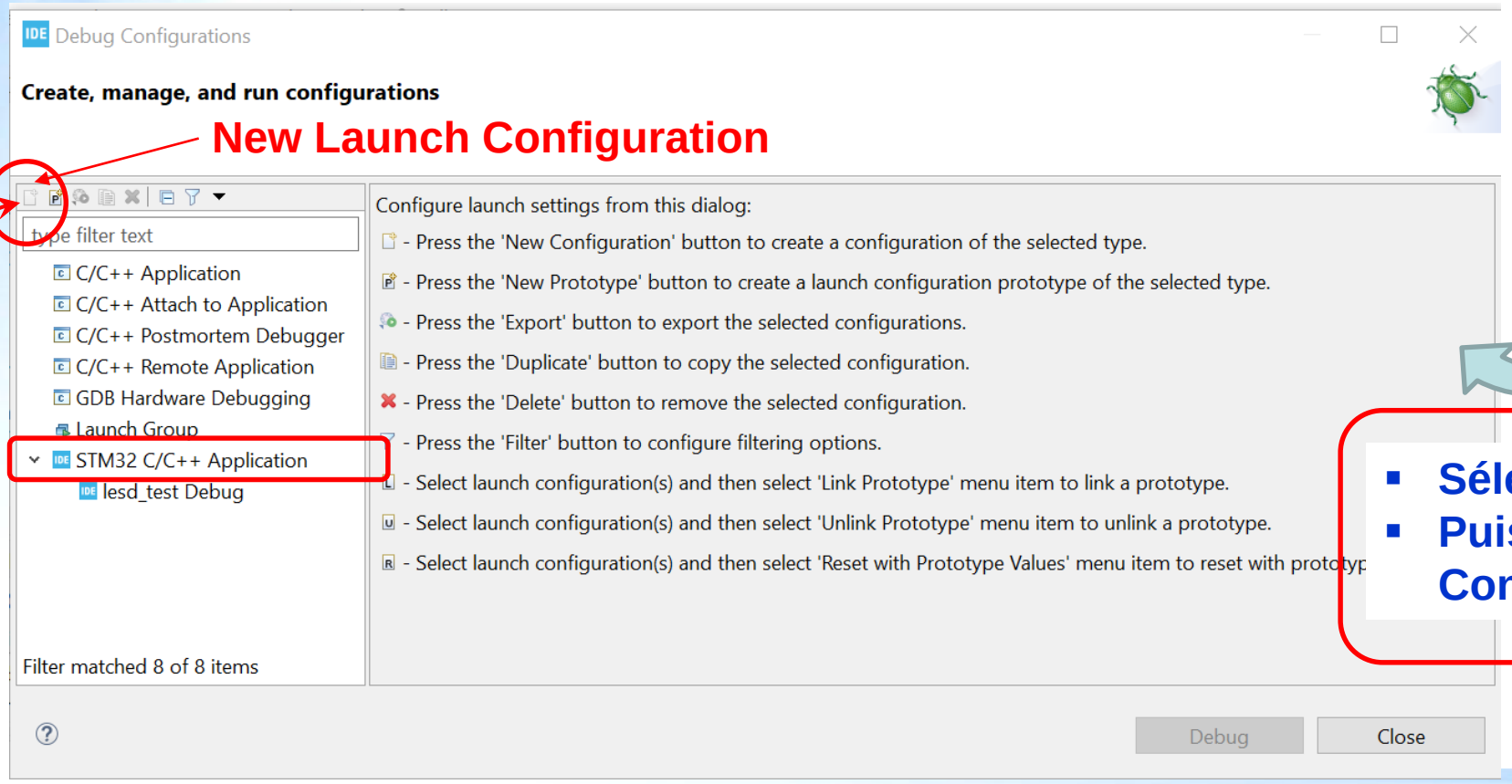
```
CDT Build Console [GPIO_LED]
arm-none-eabi-size   GPIO_LED.elf
   text    data     bss     dec     hex filename
   7752      20    1636    9408    24c0 GPIO_LED.elf
Finished building: default.size.stdout


18:11:18 Build Finished. 0 errors, 0 warnings. (took 215ms)
```

PAS D'ERREURS !!!

Configuration du débogage

La dernière étape avant de flasher le code est de créer « Debug Configuration »

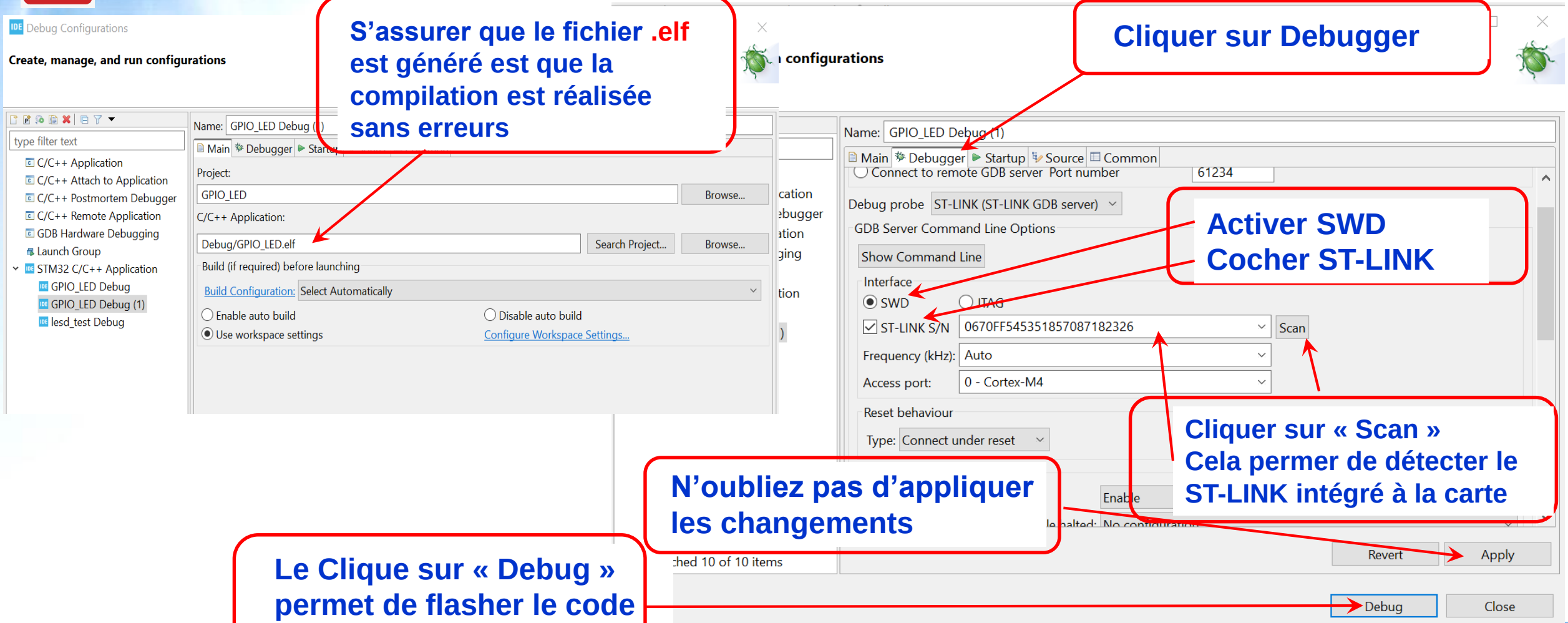




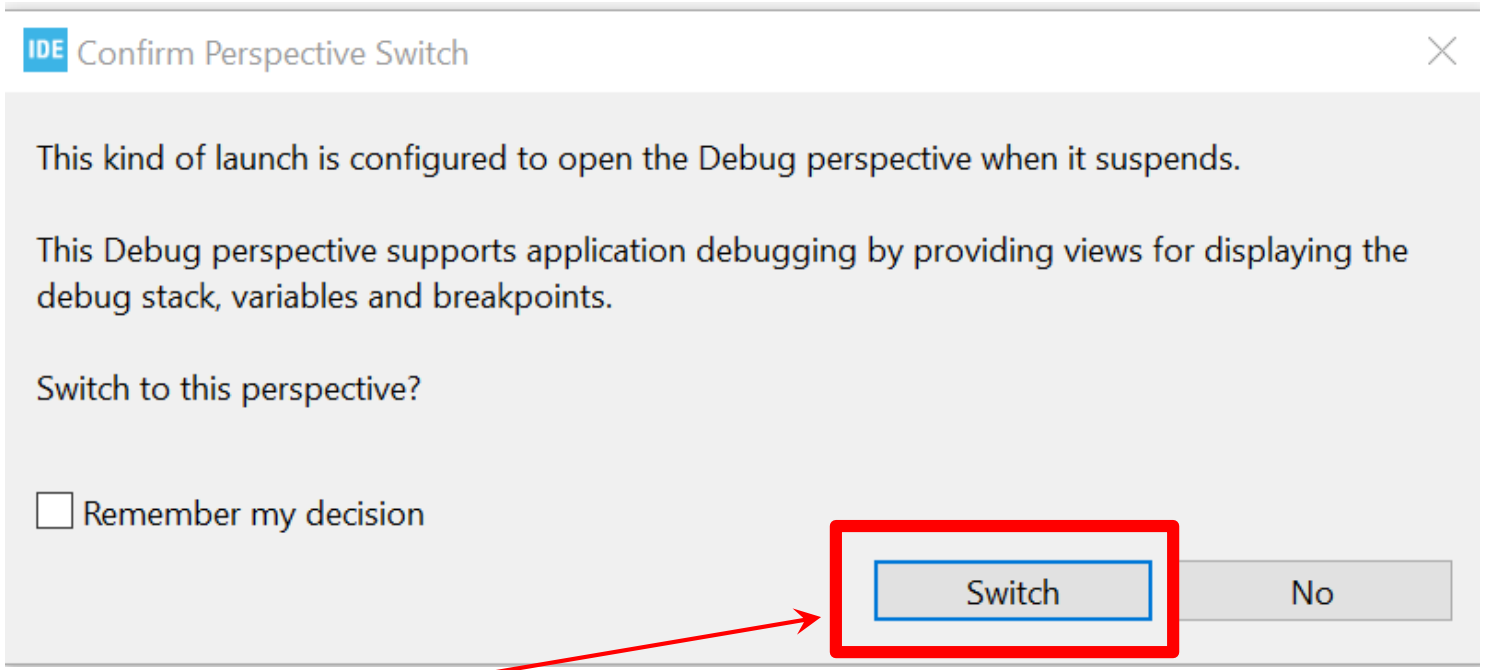
- **Sélectionner STM32C/C++Application**
- **Puis cliquer sur « New Launch Configuration »**

Débogage: flash du code

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



Débogage: flash du code

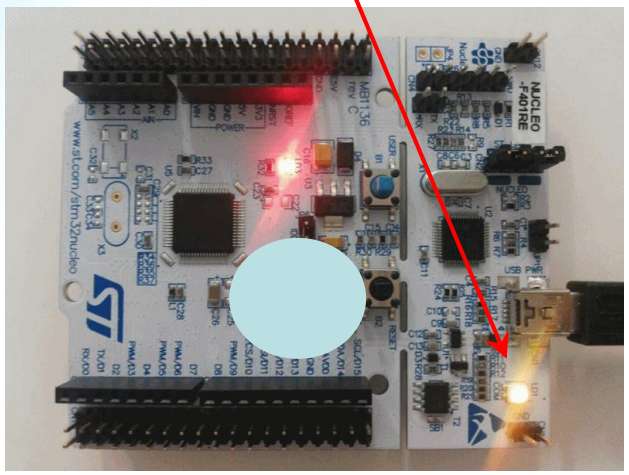


Le Cliquer sur « Switch »

Débogage: flash du code

Quand l'étape de Débogage est terminée vous apercevez:

- **Un message « Download verified successfully**
- **Le programme s'arrête à la ligne 75 (HAL_Init en surbrillance verte)**
- **La LED de la carte ST-LINK clignote vert/rouge**

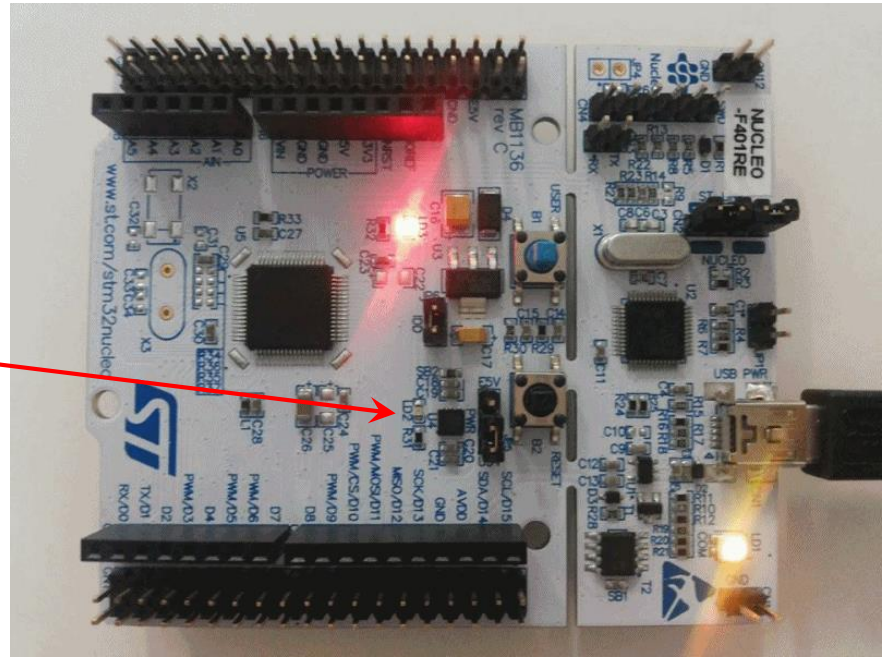


Câble mini USB V3

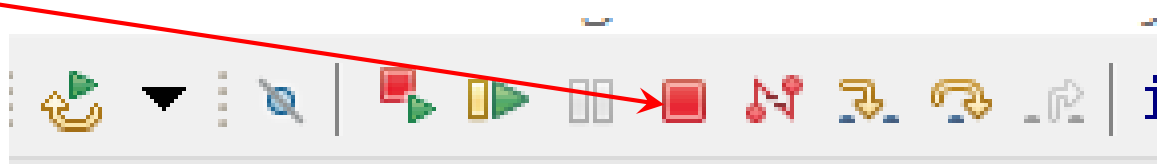
Le programme est maintenant à l'étape HAL_Init
Cette fonction n'est pas encore exécutée

Débogage: flash du code

Cliquer enfin sur ce bouton RESUME en VERT
Cela permet d'exécuter la suite du programme, et en particulier le programme du clignotement de la LED verte à une fréquence de 0.5Hz (1sec ON et 1s OFF).



Vous pouvez STOPPER le debogage en appuyant sur le bouton STOP)
Cela veut dire que le Débogage est terminé mais le programme est en exécution sur le microcontrôleur



Points d'arrêt

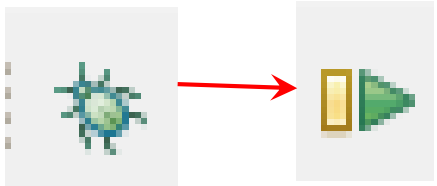
On peut ajouter des points d'arrêt dans le programme (BREAK POINTS)
On ajoute un point d'arrêt en double-cliquant sur la ligne (Exemple: ligne 104)

```
 93   /* USER CODE END 2 */
 94
 95   /* Infinite loop */
 96   /* USER CODE BEGIN WHILE */
 97   while (1)
 98   {
 99     /* USER CODE END WHILE */
100
101     /* USER CODE BEGIN 3 */
102       HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); /* Mettre la
103       HAL_Delay(1000); /* Temporistaion de 1000ms=1s */
104       HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); /* Mettre
105       HAL_Delay(1000); /* Temporistaion de 1000ms=1s */
106   }
107   /* USER CODE END 3 */
108 }
109
110 /**
```

- "Breakpoint" est une fonctionnalité de débogage qui vous permet de suspendre l'exécution de votre programme sur une ligne de code spécifique pendant le débogage.
- Lorsque vous définissez un point d'arrêt, le programme s'exécutera jusqu'à ce qu'il atteigne cette ligne, puis il se mettra en pause, vous permettant d'inspecter l'état du programme, ses variables et d'autres informations à ce stade.
- Il s'agit d'un outil indispensable pour déboguer et analyser le comportement de votre application sur un microcontrôleur STM32.

Ce symbole indique qu'il y a un point d'arrêt à la ligne 104

Cliquer de nouveau sur DEBUG puis sur RESUME



Une autre façon de programmer le Clignotement de la LED

Utiliser TogglePin:

HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin)

Dans STM32CubeIDE, le terme « **Toggle Pin** » fait généralement référence à l'action de changer l'état d'une broche numérique sur un microcontrôleur STM32 de haut (niveau logique 1) à bas (niveau logique 0) ou vice versa (**Basculer l'état**).
Le basculement d'une broche est une opération courante dans les systèmes embarqués où vous devez allumer ou éteindre un appareil connecté, tel qu'une LED, un moteur ou un capteur.

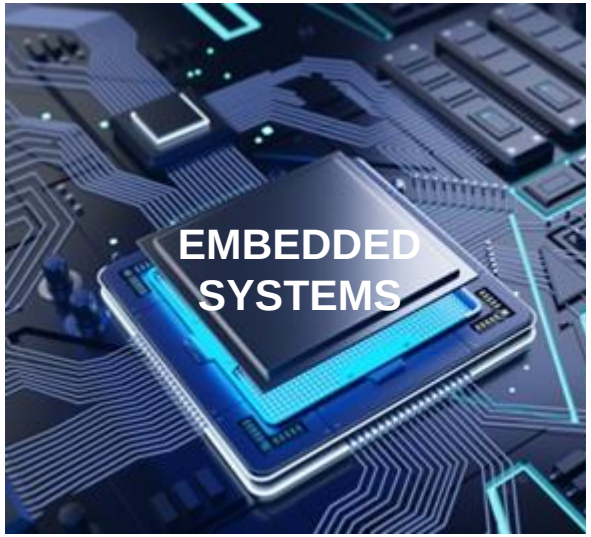
```
94
95    /* Infinite loop */
96    /* USER CODE BEGIN WHILE */
97    while (1)
98    {
99      /* USER CODE END WHILE */
100
101     /* USER CODE BEGIN 3 */
102
103       HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
104       HAL_Delay(1000); /* Temporistaion de 1000ms=1s */
105
106   }
107   /* USER CODE END 3 */
108 }
```

La Pin

Le port

Essayez de changer la fréquence de clignotement !!!

THAT'S ALL!



FIN !
Questions ?