TUTORIEL LANGAGE C POUR STM32

L'objectif de ce tutoriel est de présenter des exemples de fonctionnalités mises en œuvre en langage C sur des microcontrôleurs STM32 de chez ST Microelectronics via le logiciel STM32CubeIDE.

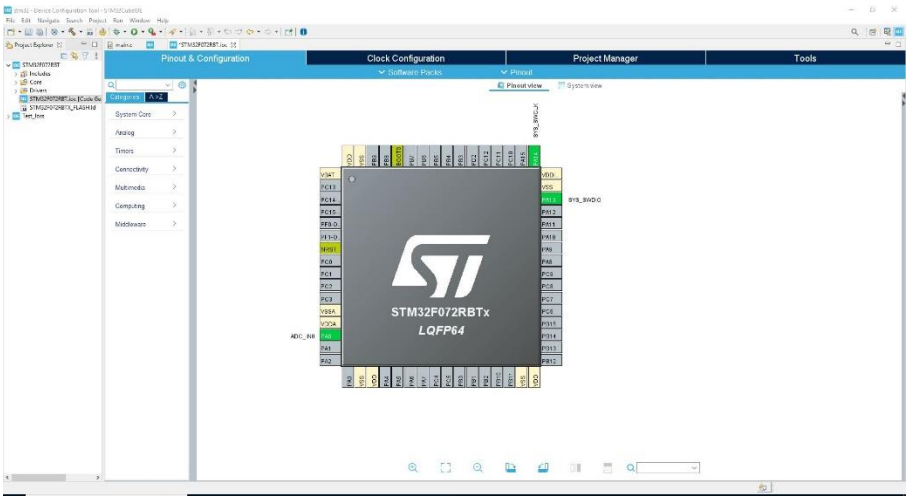
Les instructions utilisées en langage C pour programmer les microcontrôleurs STM32 seront principalement issues de la bibliothèque HAL. Cette bibliothèque HAL est accessible par défaut lors de la programmation des microcontrôleurs STM32.

Table des matières

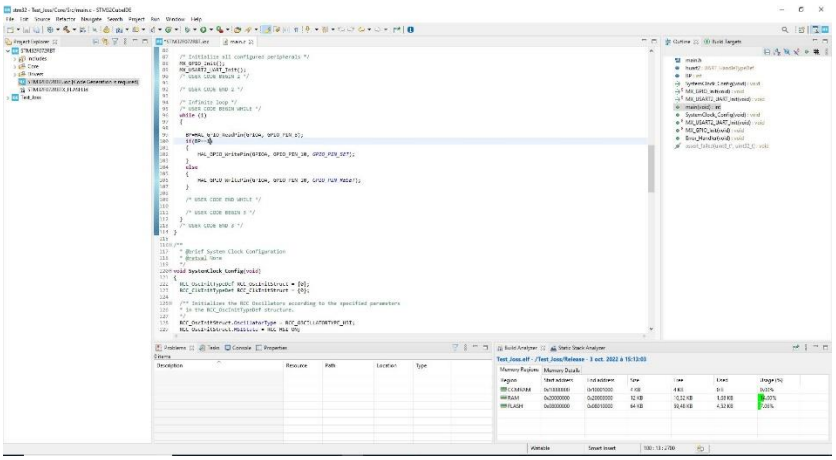
I. PRESENTATION DE L'INTERFACE

Le logiciel STM32Cube IDE regroupe plusieurs perspectives. Ces perspectives permettent les actions suivantes :

- Création d'un projet STM32 et choix du microcontrôleur.
- Initialisation des broches du microcontrôleur et génération du code d'initialisation via la perspective CubeMX



- Programmation du microcontrôleur

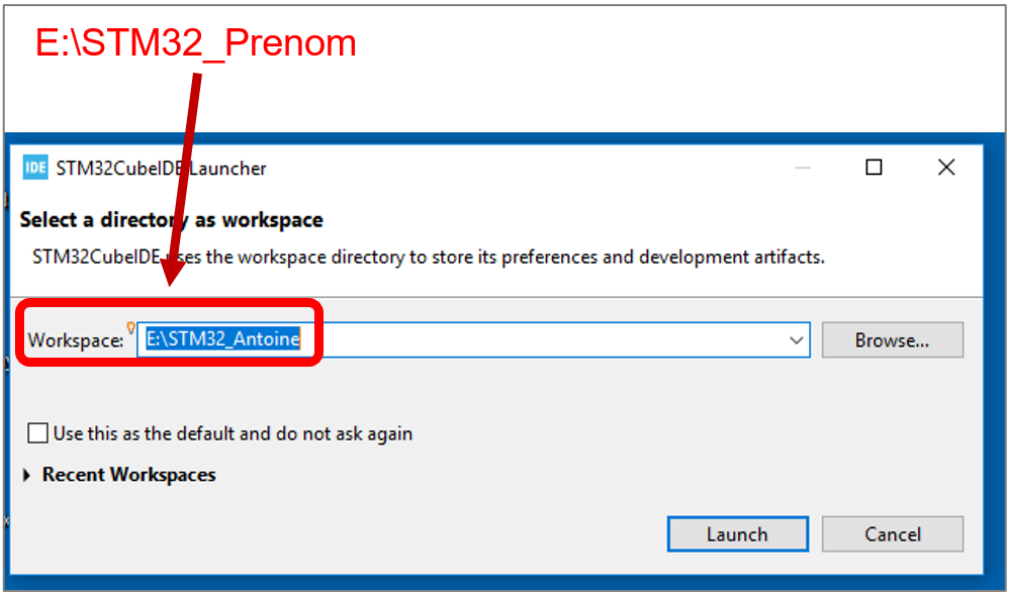


- Débuggage du programme (optionnel): Debug Mode

1. Définition de l'espace de travail

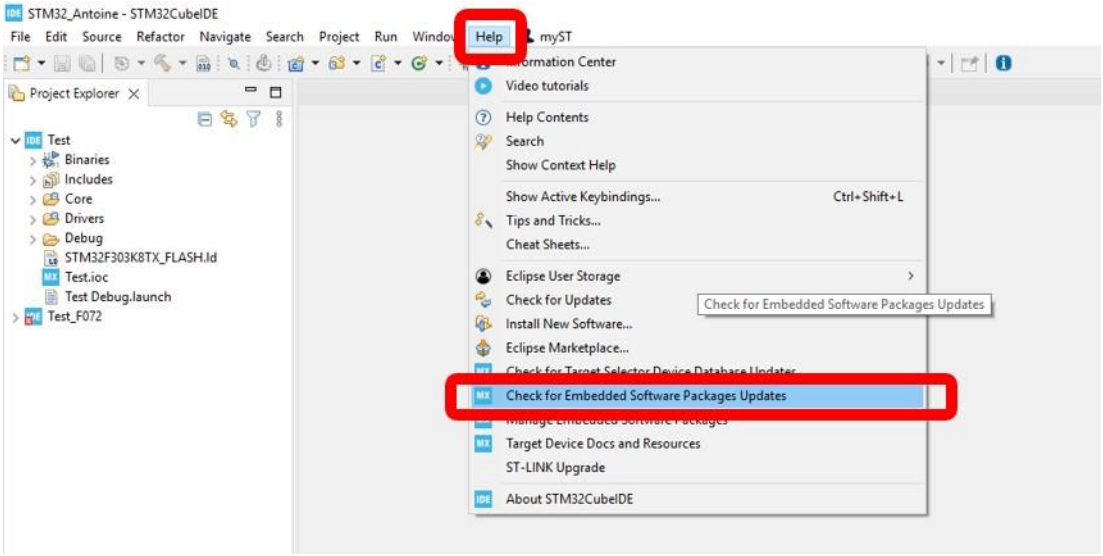
- Double-cliquer sur l'icône du logiciel STM32CubeIDE disponible sur le bureau du PC

Une fenêtre apparait afin de définir l'espace de travail. Il est absolument indispensable que votre espace de travail soit correctement défini. Il vous est demandé de créer dans le disque E : et de créer un dossier « E:\STM32_Prenom » et **d'utiliser toujours ce dossier comme espace de travail.**

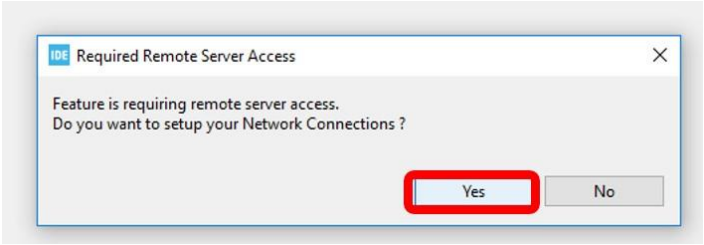


2. Définir le répertoire pour les driver

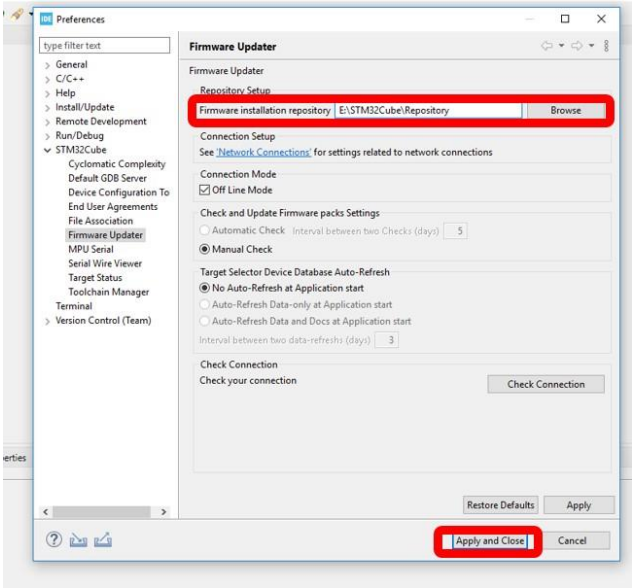
- Cliquer sur Help => Check for Embedded Software Packages Updates



- Cliquer sur Yes

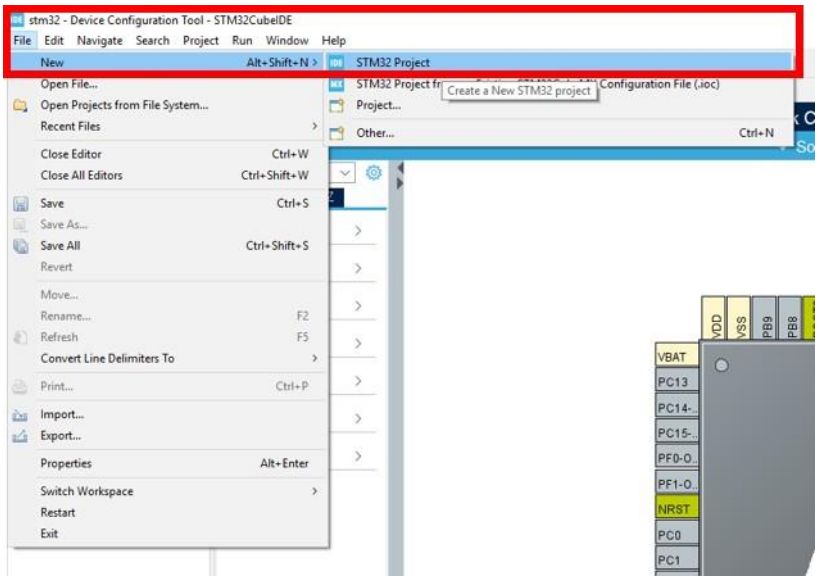


- Définir le répertoire « Firmware installation repository » dans le répertoire « E:\STM32Cube\Repository » puis cliquer sur « Apply and Close »

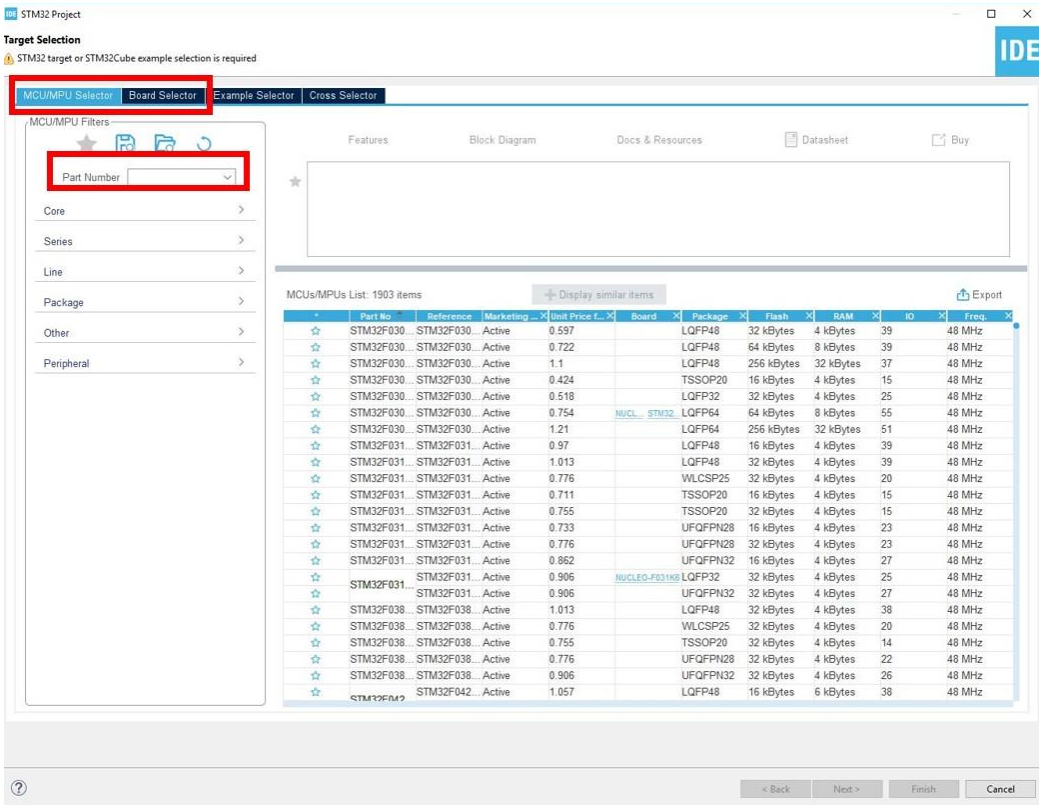


3. Création d'un projet

- Pour créer un projet, cliquer sur File => New => STM32 Project

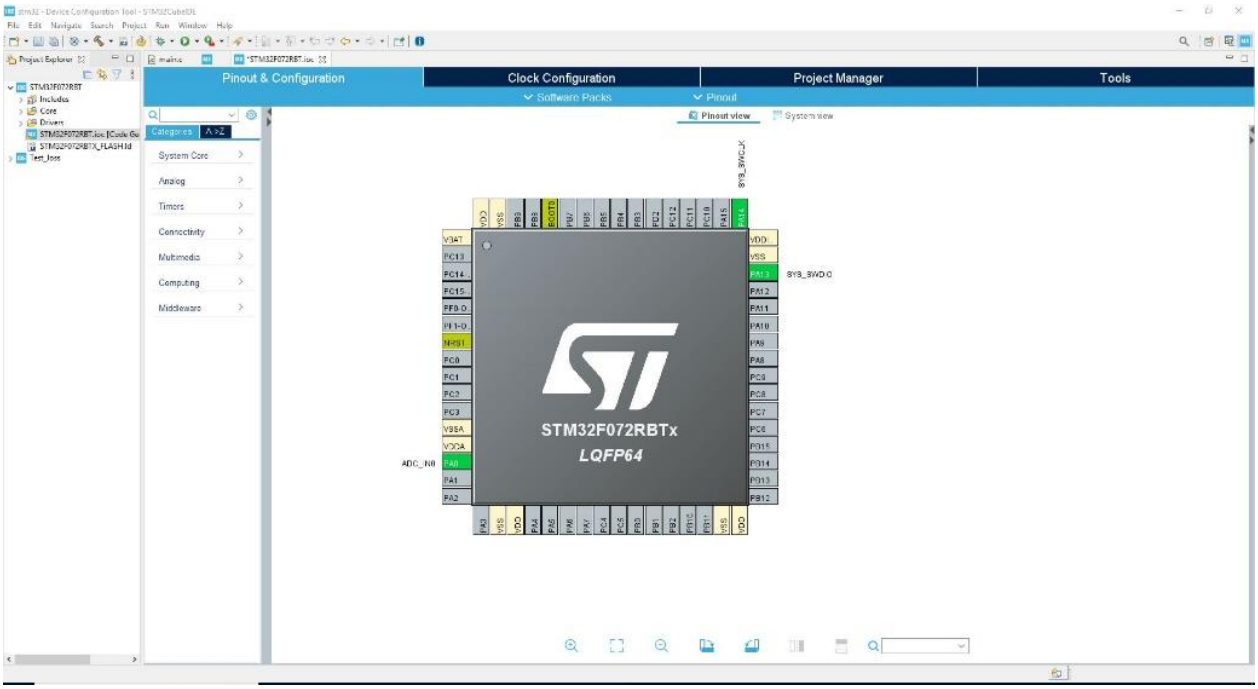


- Recherche votre microcontrôleur (MCU selector) ou carte de développement utilisé (Board Selector)



4. Initialisation des broches et création du code

Lors de la création d'un nouveau projet, la perspective CubeMX s'ouvre par défaut. Cette perspective permet une visualisation du microcontrôleur de travail et d'initialiser les broches. La perspective CubeMX est associée au fichier *.ioc.

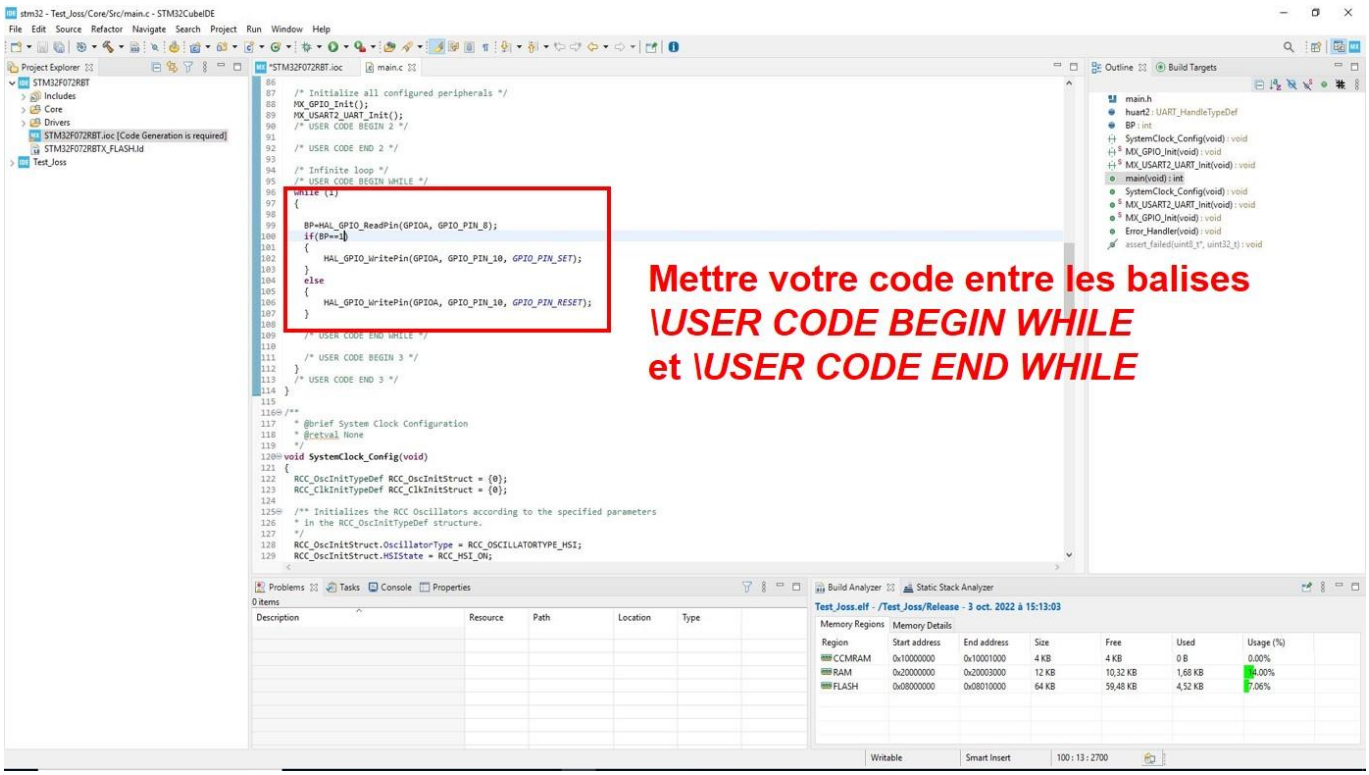


Dans l'arborescence de projet (à gauche), double-cliquer sur le fichier *.ioc permet de réouvrir la perspective CubeMX de votre projet.
Lors de la sauvegarde de votre fichier *ioc, une fenêtre proposant la génération du code apparait.

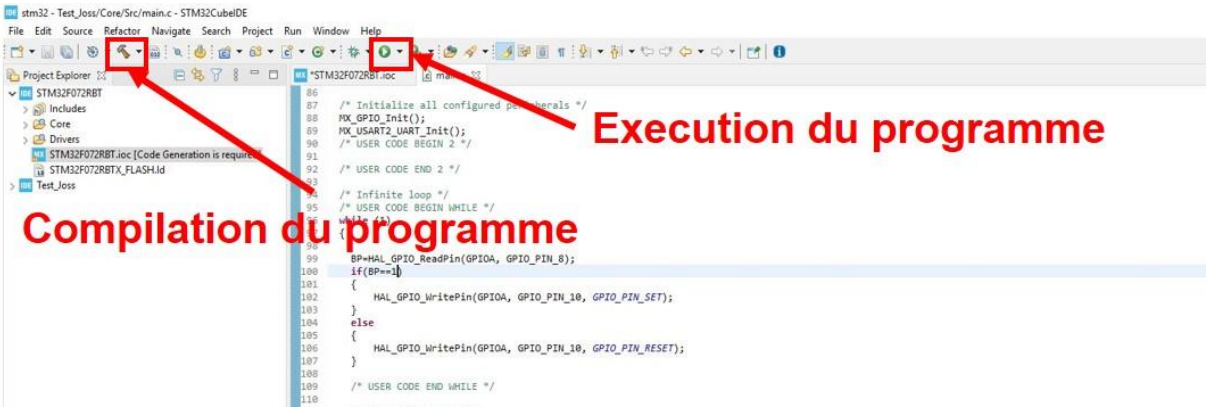
5. Programmation en C

La programme en langage C s'effectue dans le fichier main.c accessible dans l'arborescence de votre projet dans « Core => Src => main.c ». Par défaut, ce fichier main.c s'ouvre après la génération du code par CubeMX.

Vos lignes de code de votre programme doivent toujours être ajoutées entre les balises \USER CODE BEGIN WHILE et \USER CODE END WHILE



- Compiler votre programme en cliquant sur le marteau en sélectionnant le mode Release
- Lancer votre programme en cliquant sur l'outil run

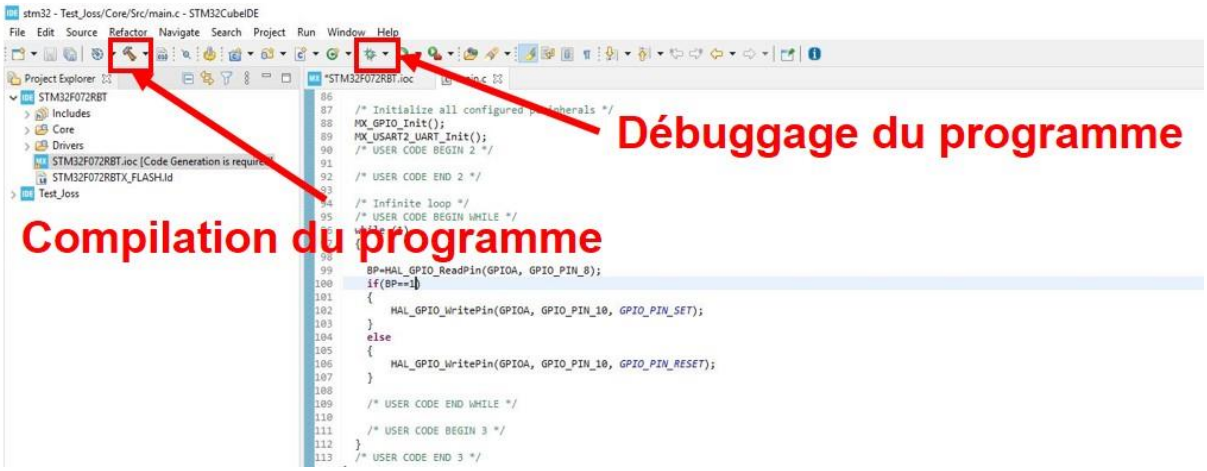


Votre programme est téléversé dans votre microcontrôleur et s'exécute.

6. Débuggage (optionnel)

Le logiciel permet de faire du débuggage. Le débuggage est un outil très utile dans le cadre de la programme d'un microcontrôleur. Il permet d'exécuter votre programme pas à pas (instruction par instruction) et de visualiser les variables.
Pour effectuer le débuggage, voici les étapes à suivre :

- Compiler votre programme en mode Debug en cliquant sur le marteau
- Lancer le débuggage en cliquant sur l'outil araignée



- Pour exécuter votre programme en pas à pas, appuyer sur le bouton ou le raccourci clavier F6.
- Pour visualiser vos variables, utiliser l'outil "Variables" disponible à droite du logiciel.

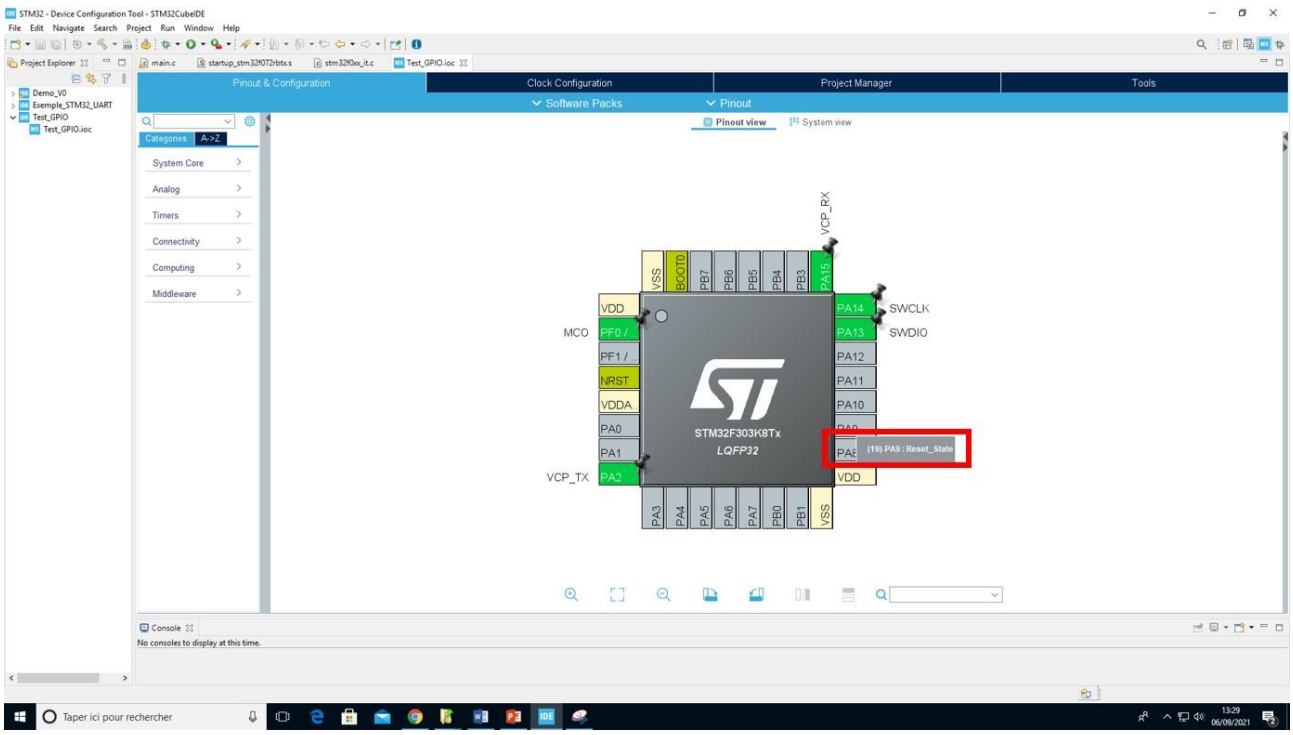
II. ENTRÉE/SORTIE NUMÉRIQUE

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à l'écriture ou la lecture de GPIO (General Purpose Input/Output traduction anglaise de : entrée/sortie numérique).

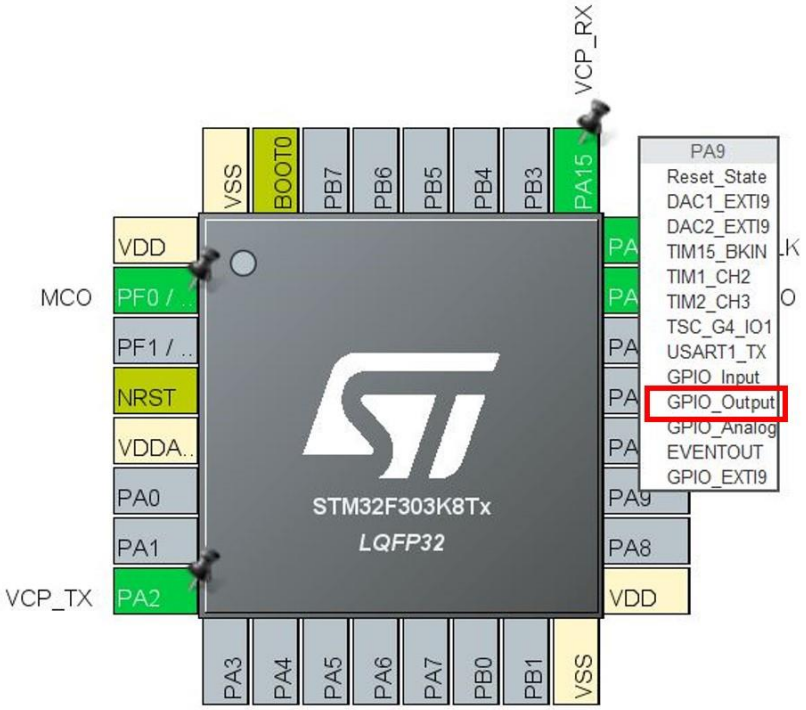
1. Initialiser une sortie numérique

L'objectif de cette partie est de présenter la méthodologie à suivre pour initialiser une sortie numérique sur le microcontrôleur STM32 à l'aide du logiciel STM32CubeMX.

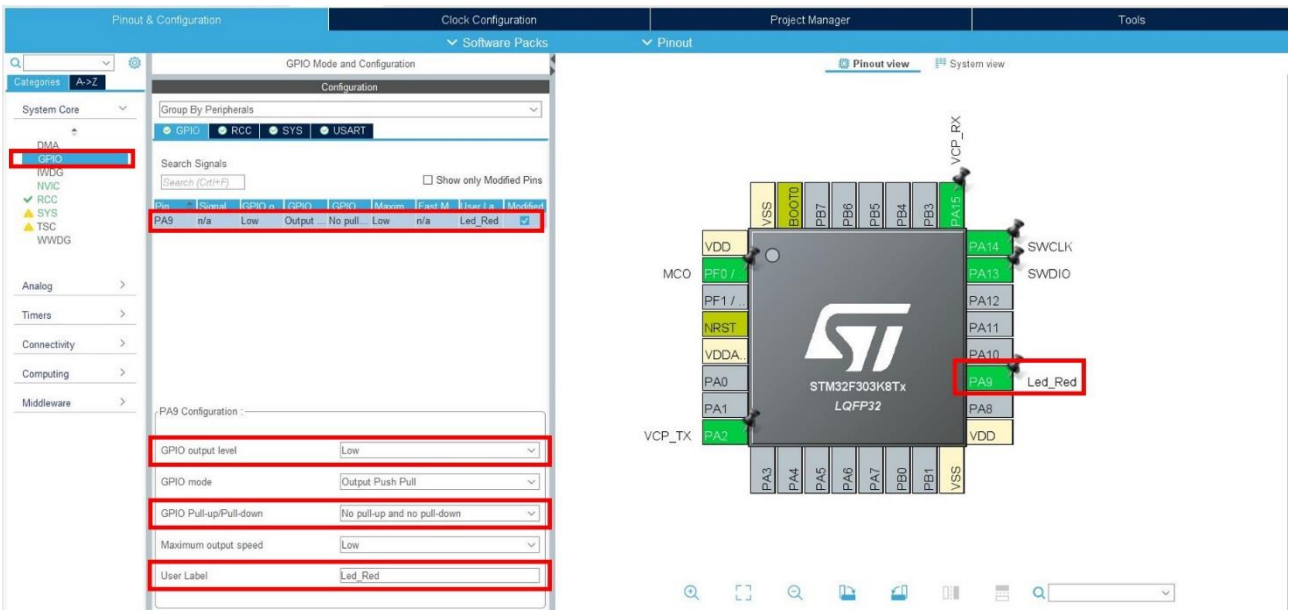
- Lancer le logiciel STM32CubeIDE
- Créer un nouveau projet lié à votre microcontrôleur STM32
- Faire un clic-droit sur la broche du microcontrôleur STM32 que vous souhaitez initialiser en sortie numérique



- Dans la liste disponible, sélectionner « GPIO_Output »



- Dans l'onglet de gauche « GPIO », les paramètres de configuration de la sortie numérique sont disponibles :
  - Niveau par défaut (GPIO Output Level)
  - Résistance de tirage interne (Pull-up, Pull-down ou Push-pull)
  - Label



- Sauvegarder votre fichier d'initialisation

2. Programmation d'une sortie numérique

L'objectif de cette partie est de présenter les instructions en langage C nécessaire à l'écriture d'une sortie numérique.
L'écriture d'une sortie numérique peut être utile pour commander une led.

- Instruction à utiliser : **HAL_GPIO_WritePin(GPIOx, GPIO_Pin, PinState);**
- Arguments de cette fonction :
  - **GPIOx** : Banque de la broche GPIO à écrire
  - **GPIO_Pin** : Numéro de la broche GPIO à écrire
  - **PinState** : Etat de la broche, *GPIO_Pin_Reset* pour 0 ou *GPIO_Pin_Set* pour 1

3. Exemple de code pour faire clignoter une LED.

On supposera que la LED est reliée à la broche PB3 du microcontrôleur.

Voici le code permettant de faire cligner la LED toutes les 200 ms :

```
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);           // Broche PB3 à l'état haut => LED ON
          HAL_Delay(200);
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);         // Broche PB3 à l'état bas => LED OFF
          HAL_Delay(200);
    /* USER CODE END WHILE */

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

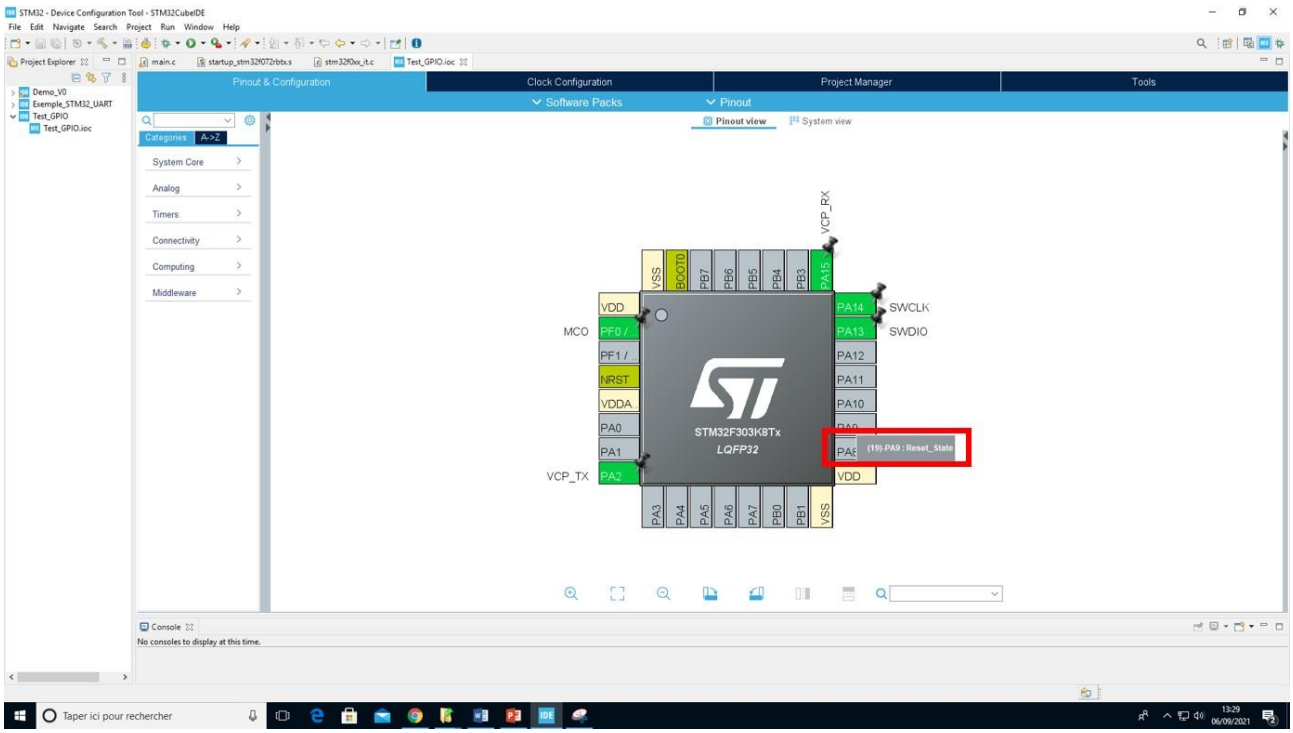
Voici un autre exemple de code permettant de faire clignoter la LED toutes les 200 ms en utilisant cette fois-ci l'instruction HAL_GPIO_TogglePin;

```
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
          HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3); // Inversion de l'état de la broche PB3
          HAL_Delay(200);
    /* USER CODE END WHILE */

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

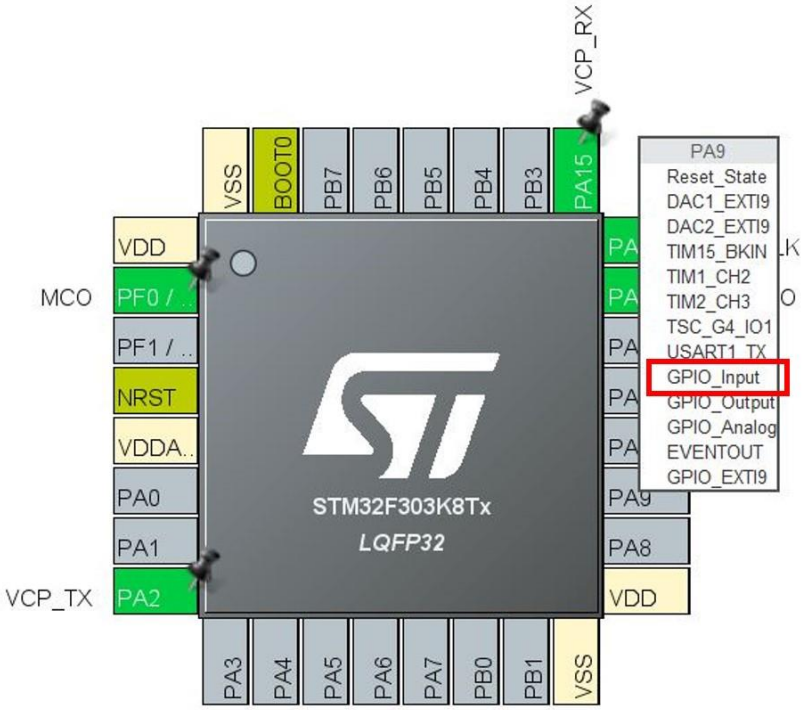
4. Initialiser une entrée numérique

L'objectif de cette partie est de présenter la méthodologie à suivre pour initialiser une entrée numérique sur le microcontrôleur STM32 à l'aide du logiciel STM32CubeMX.

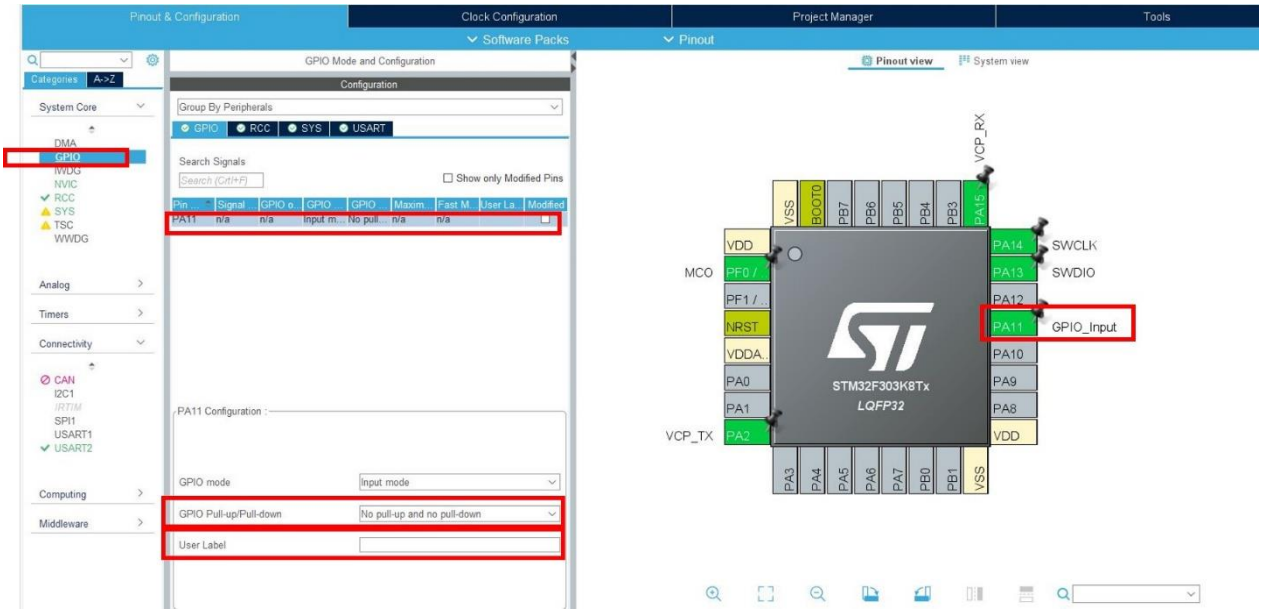
- Lancer le logiciel STM32CubeIDE
- Créer un nouveau projet lié à votre microcontrôleur STM32
- Faire un clic-droit sur la broche du microcontrôleur STM32 que vous souhaitez initialiser en entrée numérique



- Dans la liste disponible, sélectionner « GPIO_Intput »



- Dans l'onglet de gauche « GPIO », les paramètres de configuration de l'entrée numérique sont disponibles :
  - Résistance de tirage interne (Pull-up, Pull-down ou Push-pull)
  - Label



- Sauvegarder votre fichier d'initialisation

5. Programmation d'une entrée numérique

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à la lecture d'une entrée numérique.
La lecture d'une entrée numérique peut être utile pour déterminer l'état d'un bouton-poussoir (appui ou non).

- Instruction à utiliser : **HAL_GPIO_ReadPin(GPIOx, GPIO_Pin) ;**
- Arguments de cette fonction :
  - **GPIOx** : Banque de la broche GPIO à écrire
  - **GPIO_Pin** : Numéro de la broche GPIO à écrire
- Exemple pour lire l'état d'un bouton-poussoir sur la broche PA8 et allumer une led sur la broche PB3 :

6. Exemple de code pour commander une led en fonction d'un bouton-poussoir

L'exemple ci-dessous a pour but d'allumer une led lorsque le bouton-poussoir est appuyé sinon la led reste éteinte.
On supposera que le bouton-poussoir est associé à la broche PA8 et que la LED est associée à la broche PB3. On supposera que lorsque le bouton-poussoir est appuyé, une tension de 3V3 (état haut, '1' logique) est appliquée à la broche PA8. Lorsque le bouton-poussoir est relâché, une tension de 0V (état bas, '0' logique) est appliquée à la broche PA8. La broche PA8 est donc configurée avec une résistance de pull-down.

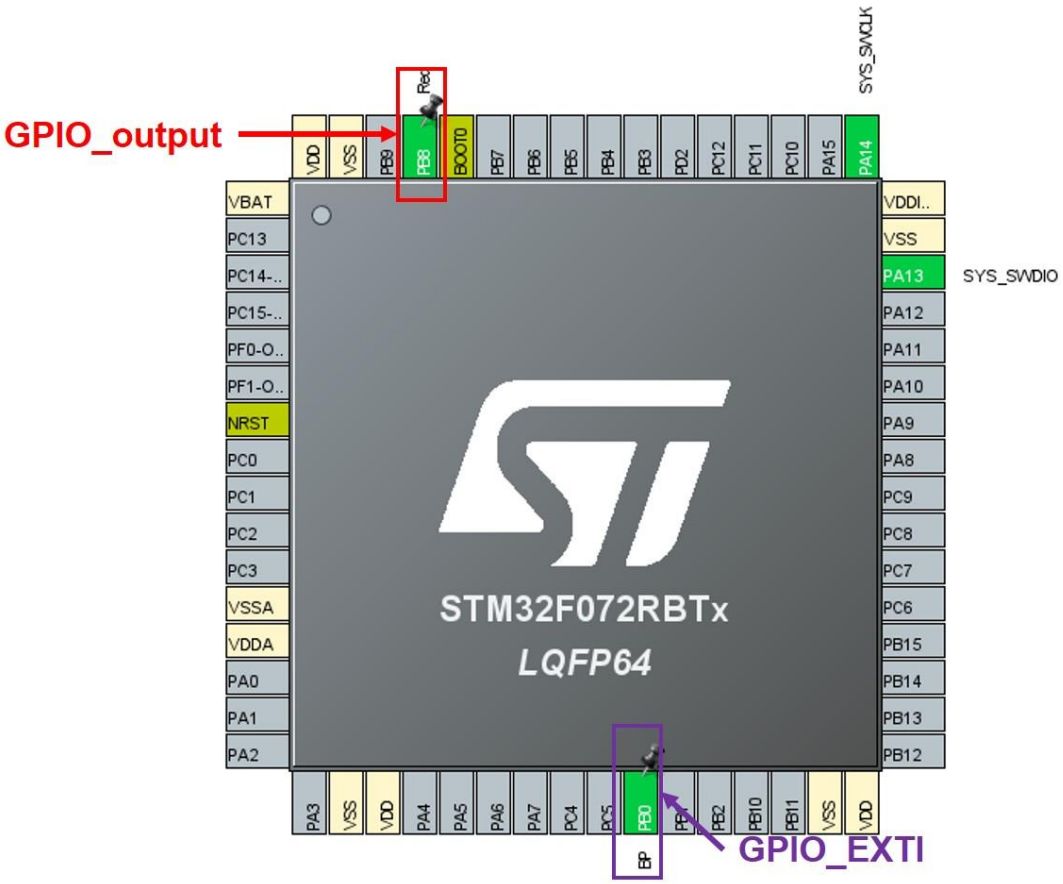
```
  /* USER CODE BEGIN 2 */
  int BP;                                    // Variable associée au bouton-poussoir
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
          BP = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8);      // Lecture de la broche PA8 associée au bouton-poussoir
          if (BP == 1)
          {
                  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);    // Broche PB3 à l'état haut => LED ON

          }
          else {
                  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET); // Broche PB3 à l'état bas => LED OFF

          }

    /* USER CODE END WHILE */

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

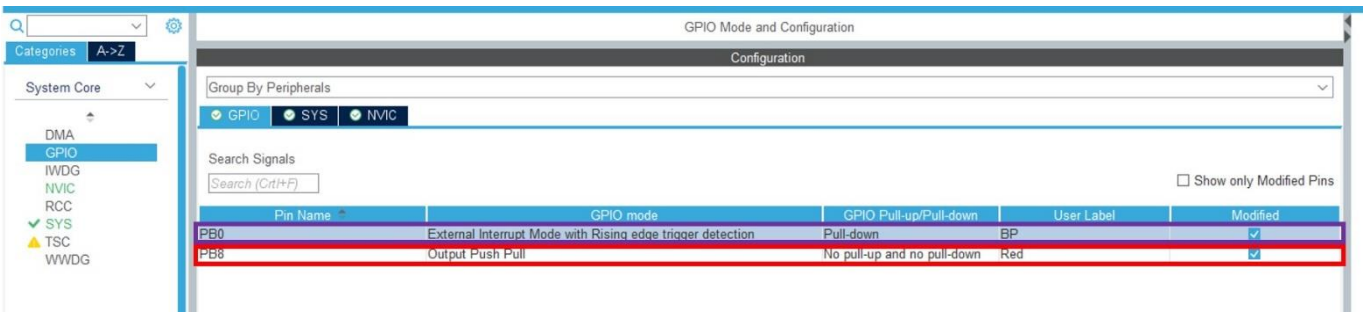
7. Gestion des interruptions pour les GPIO

L'objectif de cette partie est de présenter la mise-en-œuvre des interruptions pour les GPIO.
Une interruption est un évènement qui déclenche l'exécution immédiate d'un programme. Les interruptions sont très utiles pour permettre la détection d'évènement prioritaire (réception de données, appui sur un bouton-poussoir, …)
Dans cet exemple, l'objectif est d'allumer ou d'éteindre une led à l'aide d'une interruption sur le bouton-poussoir.

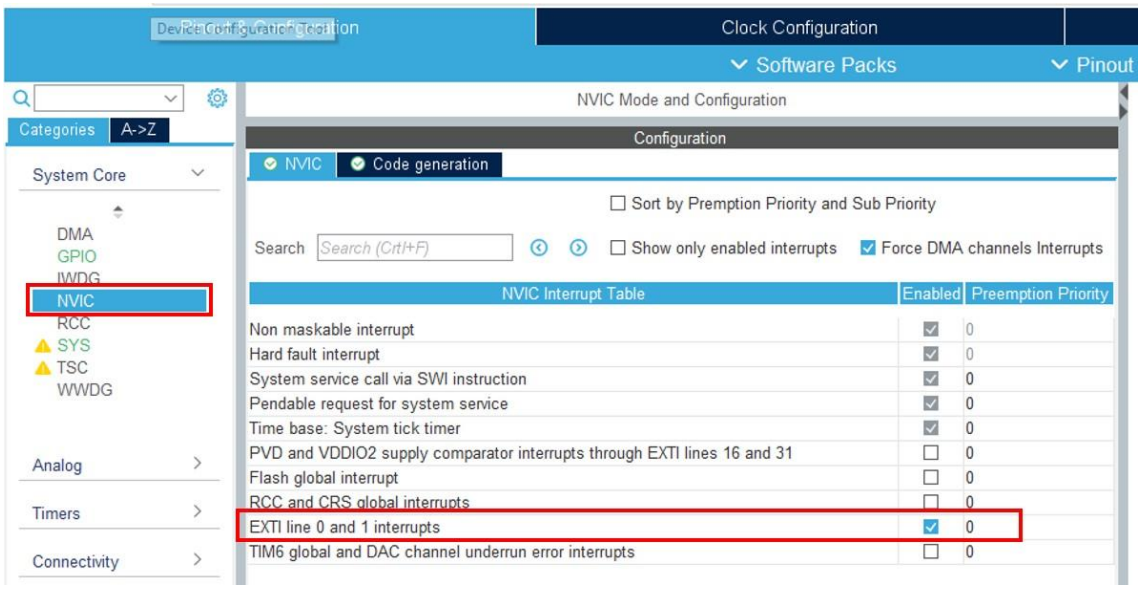
- Sur l'interface CubeMX, initialiser les broches du microcontrôleur

La broches PB8 : *GPIO_output* avec un pull-down et un état de référence à l'état haut (Led éteinte)
La broche PB0 : *GPIO_EXTI* (activation de l'interruption) en mode « *External Interrupt Mode with Rising edge trigger detection* ».

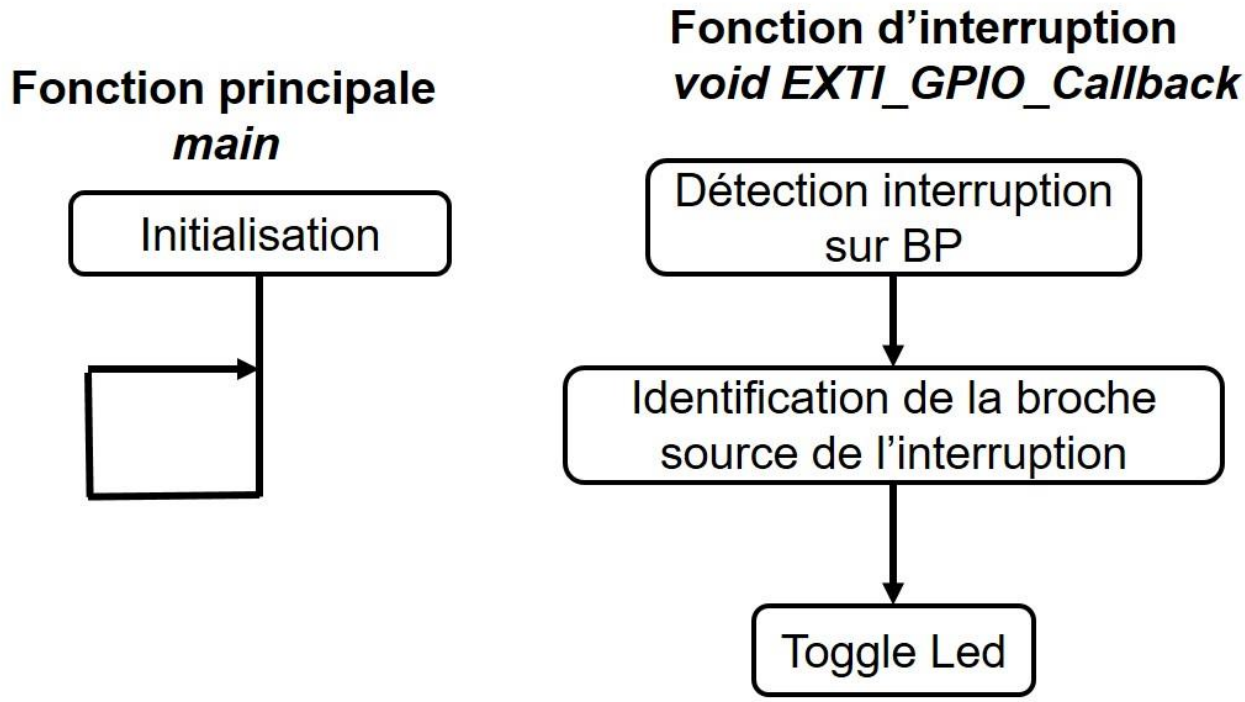




- Sur l'interface CubeMX, activer les interruptions en allant dans l'onglet « *NVIC* » et en cochant la case « *EXTI line 0 and 1 interrupt* ».



L'algorigramme ci-dessous présenta le programme à créer pour allumer ou éteindre la led en fonction de l'interruption générée par le bouton-poussoir.



La programmation consiste à entrer dans la fonction d'interruption uniquement lorsque l'utilisateur appui ou relâche le bouton-poussoir. Cette fonction d'interruption sera à programmer. Elle doit **impérativement** s'appeler : *HAL_GPIO_EXTI_Callback*
La fonction principale *main* ne fait rien. Tout se passe dans cette fonction d'interruption.
La fonction *main* ne possède pas de ligne de code particulière.
La fonction *HAL_GPIO_EXTI_Callback*

- *détecte l'appui du bouton-poussoir*
- *allume/éteint la led*

Voici les lignes de code associées

- Programmation de la fonction *EXTI_GPIO_Callback*

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback( uint16_t GPIO_Pin)
{
	if (GPIO_Pin == BP_Pin)
	{
		HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3); // Allume ou éteint la LED
	}
	else
	{

	}
}
/* USER CODE END 4 */
```

La fonction *EXTI_GPIO_Callback* est à coder entièrement entre les balises */*USER CODE BEGIN 4 */* et */* USER CODE END 4 */* située vers la ligne 150 du programme.

Pour plus d'information, consulter la vidéo suivante : https://www.youtube.com/watch?v=UtkszckecV8

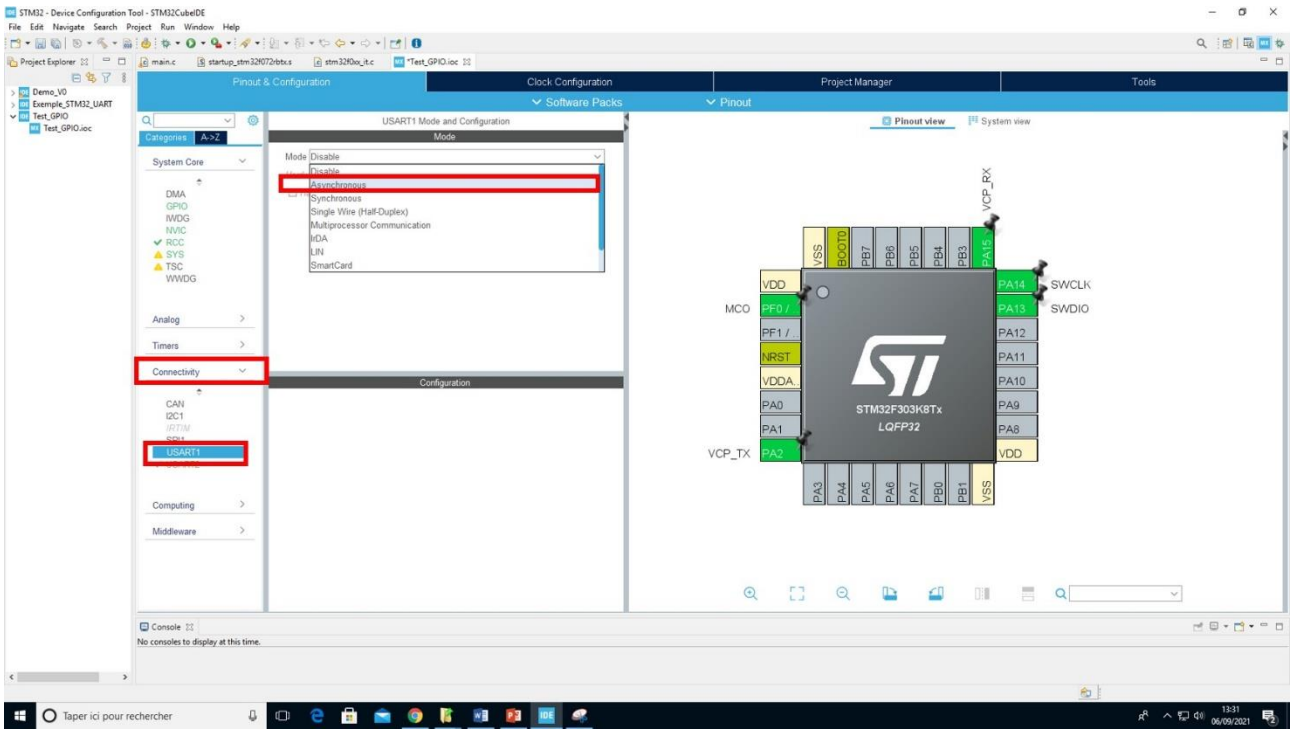
III. LIAISON SÉRIE UART

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à l'émission ou la réception de données via une liaison série de type UART.
On supposera que la liaison série a été créée et est associée à l'instance huart1.

1. Initialiser une liaison série UART

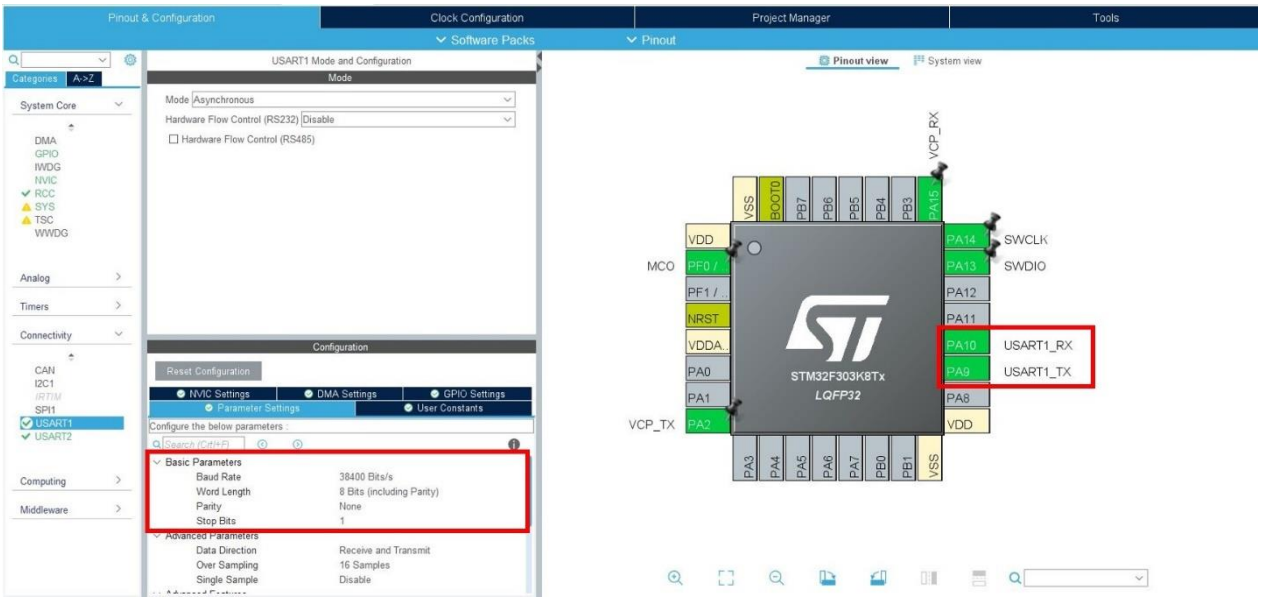
L'objectif de cette partie est de présenter la méthodologie à suivre pour initialiser une liaison série UART sur le microcontrôleur STM32 à l'aide du logiciel STM32CubeMX.

- Lancer le logiciel STM32CubeIDE
- Créer un nouveau projet lié à votre microcontrôleur STM32
- Dans l'onglet de gauche « *Connectivity* », sélectionner USART puis le mode Asynchrone



Les broches du microcontrôleur STM32 attribuée à cette liaison série UART apparaissent.

- Les paramètres de configuration de la liaison série UART sont disponibles :
  - Débit
  - Longueur de bit de donnée
  - Bit de parité
  - Bit de stop



- Sauvegarder votre fichier d'initialisation

2. Emettre une trame via une liaison UART

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à l'émission de données via une liaison série de type UART.
On supposera que la liaison série a été créée et est associée à l'instance huart1.

- Instruction à utiliser : **HAL_UART_Transmit(*huart, pData, Size, Timeout) ;**
- Arguments de cette fonction :
  - ***huart** : pointeur vers l'instance de la liaison UART à utiliser
  - **pData** : Donnée à transmettre
  - **Size** : taille en octet de la donnée à transmettre
  - **Timeout** : durée au-delà de laquelle la donnée ne sera pas transmise, valeur typique = 1000ms

3. Programmation pour transmettre une trame en UART

L'exemple ci-dessous présente les lignes de code pour transmettre la donnée « CIEL-ER » en liaison UART toute les 1s.

- Vérifier que les bibliothèques stdio.h et string.h sont bien incluses dans votre programme « main.c ». Pour cela, vérifier vers la ligne 22 de votre programme que les deux instructions *#include <stdio.h>* et *#include <string.h>* sont bien présentes. Si ce n'est pas le cas, ajouter les instructions *#include <stdio.h>* et *#include <string.h>* entre les balises indiquées ci-dessous :

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
/* USER CODE END Includes */
```

- Dans l'en-tête du programme, déclarer la variable Data_TX, tableau de 7 octets qui contient le message à transmettre.

```
/* USER CODE BEGIN PV */
uint8_t Data_TX[7];            // Déclaration de la variable Data_TX sur 7 octets
/* USER CODE END PV */
```

- Dans la fonction *main()*, initialiser la variable Data_TX avec les données.

```
  /* USER CODE BEGIN 2 */
  Data_TX[0] = 'C';             // Octet 0 de la variable Data_TX associé au caractère ASCII 'C'
  Data_TX[1] = 'I';             // Octet 1 de la variable Data_TX associé au caractère ASCII 'I'
  Data_TX[2] = 'E';             // Octet 2 de la variable Data_TX associé au caractère ASCII 'E'
  Data_TX[3] = 'L';             // Octet 3 de la variable Data_TX associé au caractère ASCII 'L'
  Data_TX[4] = '-';             // Octet 4 de la variable Data_TX associé au caractère ASCII '-'
  Data_TX[5] = 'E';             // Octet 5 de la variable Data_TX associé au caractère ASCII 'E'
```

```
Data_TX[6] = 'R';             // Octet 6 de la variable Data_TX associé au caractère ASCII 'R'
/* USER CODE END 2 */
```

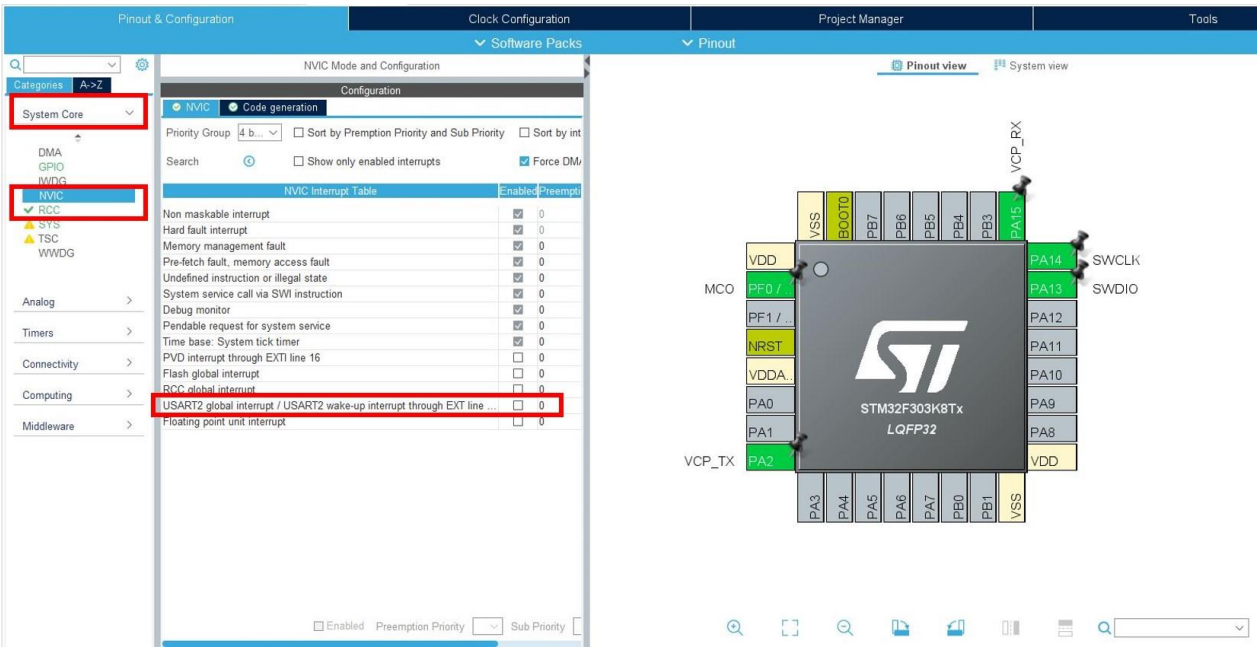
- Dans la partie boucle infinie (*while (1)*) de la fonction *main()*, transmettre la variable *Data_TX* avec les données.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
        HAL_UART_Transmit(&huart1, Data_TX, sizeof(Data_TX), 1000);    // Transmission de Data_TX en UART
        HAL_Delay(1000);                                               // Attente d'1s
  /* USER CODE END WHILE */

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

4. Recevoir une donnée via une liaison UART en mode interruption

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à la réception de données via une liaison série de type UART en mode interruption.
On supposera que la liaison série a été créée et est associée à l'instance huart1.
Le mode interruption consiste à exécuter un sous-programme lorsqu'une interruption est déclenchée. Lors d'une interruption le programme principal s'arrête là où il en est, le sous-programme (appelé *CallBack*) s'exécute puis le programme principal reprend là où il en était. Cela permet d'assurer la réception des données dès qu'elles arrivent sur la liaison UART. En mode interruption, il n'y a pas de notion de *timeout* car la réception des données n'a lieu que lorsque des données arrivent.

- Activer les interruptions sur la liaison UART en allant dans l'onglet NVIC puis USART Interrupt



5. Programmation pour recevoir une trame UART en mode interruption

L'exemple suivant présente les lignes de code pour recevoir une trame en UART en mode interruption. On supposera que la trame à recevoir est de 9 octets.
Voici la méthodologie à suivre :

- Dans l'en-tête du programme, déclarer la variable Data_RX, tableau de 9 octets qui contient le message à transmettre.

```
/* USER CODE BEGIN PV */
uint8_t Data_RX[9];            // Déclaration de la variable Data_RX sur 9 octets
/* USER CODE END PV */
```

- Dans la fonction main(), activer la réception UART par interruption.

```
/* USER CODE BEGIN 2 */
HAL_UART_Receive_IT(&huart1, Data_RX, 9);    // Activation de la réception UART par inter-
ruption
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

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

- Instruction à utiliser : **HAL_UART_Receive_IT(*huart, pData, Size) ;**
- Arguments de cette fonction :
  - ***huart** : pointeur vers l'instance de la liaison UART à utiliser
  - **pData** : Variable où stocker la donnée reçue
  - **Size** : taille en octet de la donnée à recevoir

- Définir la fonction d'interruption UART :

```
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback (UART_HandleTypeDef *huart)
{
      HAL_UART_Receive_IT(&huart1, Data_RX, 9); // Activation de la réception UART par in-
terruption
}
/* USER CODE END 4 */
```

La fonction *HAL_UART_RxCpltCallback* est à coder entièrement entre les balises */*USER CODE BEGIN 4 */* et */* USER CODE END 4 */*.

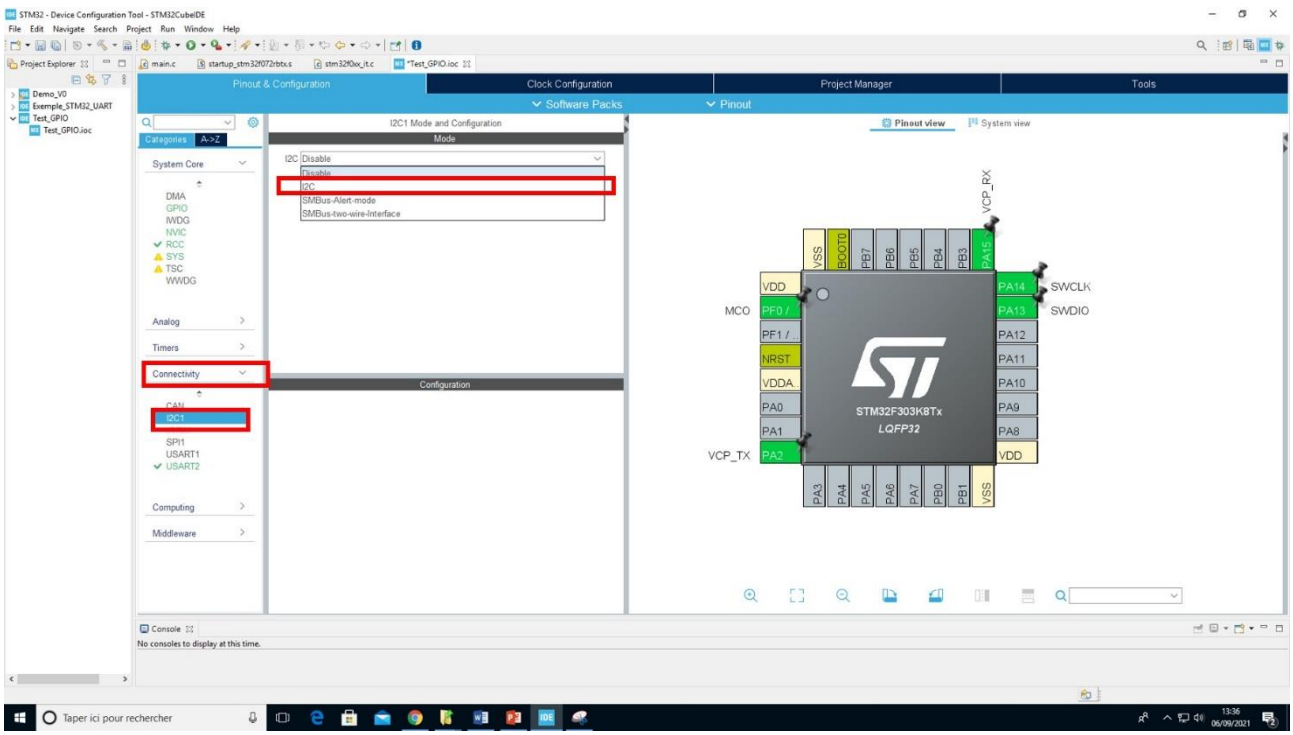
IV. BUS I2C

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à l'émission et la réception de données via une liaison série de type I2C.
On supposera que le bus I2C a été créé et est associé à l'instance hi2c1.
On supposera également que le microcontrôleur STM32 est le maître (master) de la communication I2C et que les périphériques sont les esclaves (slave).

1. Initialiser un bus I2C

L'objectif de cette partie est de présenter la méthodologie à suivre pour initialiser un bus I2C sur le microcontrôleur STM32 à l'aide du logiciel STM32CubeMX.

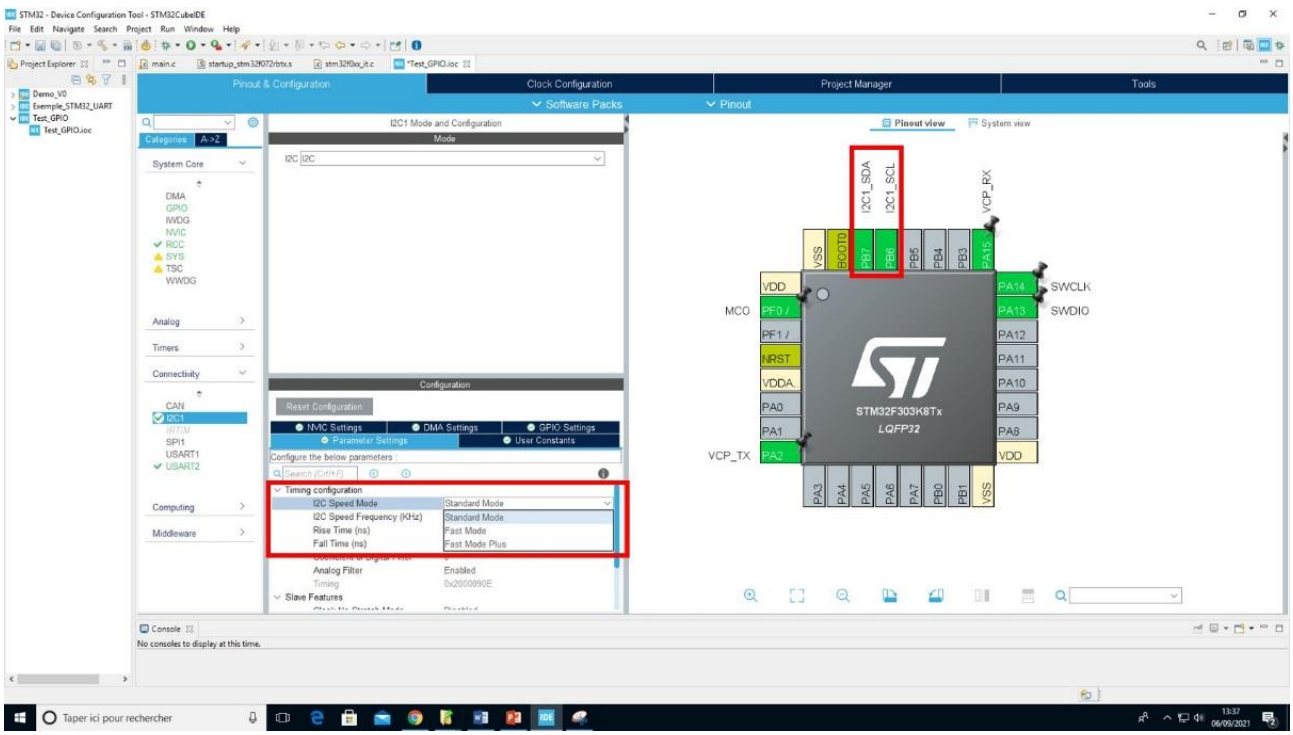
- Lancer le logiciel STM32CubeIDE
- Créer un nouveau projet lié à votre microcontrôleur STM32
- Dans l'onglet de gauche « Connectivity », sélectionner I2C puis le mode I2C



Les broches du microcontrôleur STM32 attribuées à ce bus I2C apparaissent.

- Les paramètres de configuration du bus I2C sont disponibles :
  - Mode I2C (Standard ou High Speed)

Le mode I2C définit le débit du bus I2C.



- Sauvegarder votre fichier d'initialisation

2. Emettre une donnée via un bus I2C

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à l'émission de données via une liaison série de type I2C.
On supposera que le bus I2C a été créé et est associé à l'instance hi2c1.
L'émission d'une donnée via un bus I2C peut être nécessaire pour configurer un périphérique.

- Instruction à utiliser : **HAL_I2C_Master_Transmit(*hi2c, DevAddress, *pData, Size, Timeout) ;**
- Arguments de cette fonction :
  - ***hi2c** : pointeur vers instance du bus I2C à utiliser
    DevAddress: Adresse du périphérique I2C destinataire de la donnée
  - ***pData** : pointeur vers la donnée à transmettre
  - **Size** : taille en octet de la donnée à transmettre
  - **Timeout** : durée au-delà de laquelle la donnée ne sera pas transmise, valeur typique = 50ms
- Retour de la fonction HAL_I2C_Master_Transmit : HAL_OK si la transmission est réussie

3. Exemple pour transmettre une trame en I2C

L'exemple ci-dessous présente les lignes de code pour transmettre une trame de 1 octet (0x43) en I2C avec confirmation visuelle de la transmission est OK. On supposera qu'une LED est associée à la broche PA9 et que la liaison I2C est initialisée et associée à l'instance hi2c1.

- Dans l'en-tête du programme, déclarer les variables utiles :

```
/* USER CODE BEGIN PV */
  uint8_t PERIPH_ADDR;    //Adresse I2C du périphérique sur 7 bits
  uint8_t CONFIG_REG;             //Registre cible
  uint8_t Data_TX;          //Données à transmettre
  HAL_StatusTypeDef ret ;

/* USER CODE END PV */
```

- Dans le début de la fonction main(), intialiser les variables utiles :

```
  /* USER CODE BEGIN 2 */
  PERIPH_ADDR = 0x18 <<1  ;     //Adresse I2C du périphérique sur 7 bits
  CONFIG_REG = 0x01;              //Registre cible
  Data_TX = 0x43;           //Données à transmettre
  /* USER CODE END 2 */
```

- Dans la partie boucle infinie (**while** (1)), programmer l'émission I2C et la confirmation visuelle.

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    ret = HAL_I2C_Master_Transmit(&hi2c1, PERIPH_ADDR, &CONFIG_REG, 1, 1000); //Transmis-
sion en I2C

//Led ON si transmission I2C OK
  if (ret !=HAL_OK){
       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET); ); //Led ON
       }
  else {
       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET); //Led OFF
       }
  HAL_Delay(500);
    /* USER CODE END WHILE */

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

4. Recevoir une donnée via un bus I2C

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à la réception de données via un bus I2C.
On supposera que le bus I2C a été créé et est associé à l'instance hi2c1.
L'émission d'une donnée via un bus I2C peut être nécessaire pour récupérer les données provenant d'un périphérique.

- Instruction à utiliser : **HAL_I2C_Master_Receive(*hi2c, DevAddress, pData, Size, Timeout) ;**
- Arguments de cette fonction :
  - ***hi2c** : pointeur vers instance du bus I2C à utiliser
  - **DevAddress**: Adresse du périphérique I2C destinataire de la donnée
  - **pData** : Variable où stocker la donnée à recevoir
  - **Size** : taille en octet de la donnée à recevoir
  - **Timeout** : durée au-delà de laquelle la donnée ne sera pas reçue, valeur typique = 50ms

5. Exemple pour recevoir une trame en I2C

Voici un exemple afin de recevoir une trame de 2 octets d'un registre cible (0x23).

- Dans l'en-tête du programme, déclarer les variables utiles :

```
/* USER CODE BEGIN PV */
uint8_t PERIPH_ADDR;      //Adresse I2C du périphérique sur 7 bits
uint8_t CONFIG_REG;         //Registre cible
uint8_t Data_RX[2];         //Données à recevoir
HAL_StatusTypeDef ret ;

/* USER CODE END PV */
```

- Dans le début de la fonction **main**(), initialiser les variables utiles :

```
/* USER CODE BEGIN 2 */
PERIPH_ADDR = 0x18 <<1    ;      //Adresse I2C du périphérique sur 7 bits
CONFIG_REG = 0x023;         //Registre cible

/* USER CODE END 2 */
```

- Dans la partie boucle infinie (**while** (1)), programmer la réception I2C.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  ret = HAL_I2C_Master_Transmit(&hi2c1, PERIPH_ADDR, &CONFIG_REG, 1, 1000);   //Transmission en I2C
  HAL_I2C_Master_Receive(&hi2c1, PERIPH_ADDR, Data_RX, sizeof(Data_RX), 1000) //Réception I2C
  HAL_Delay(500);
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
}
```

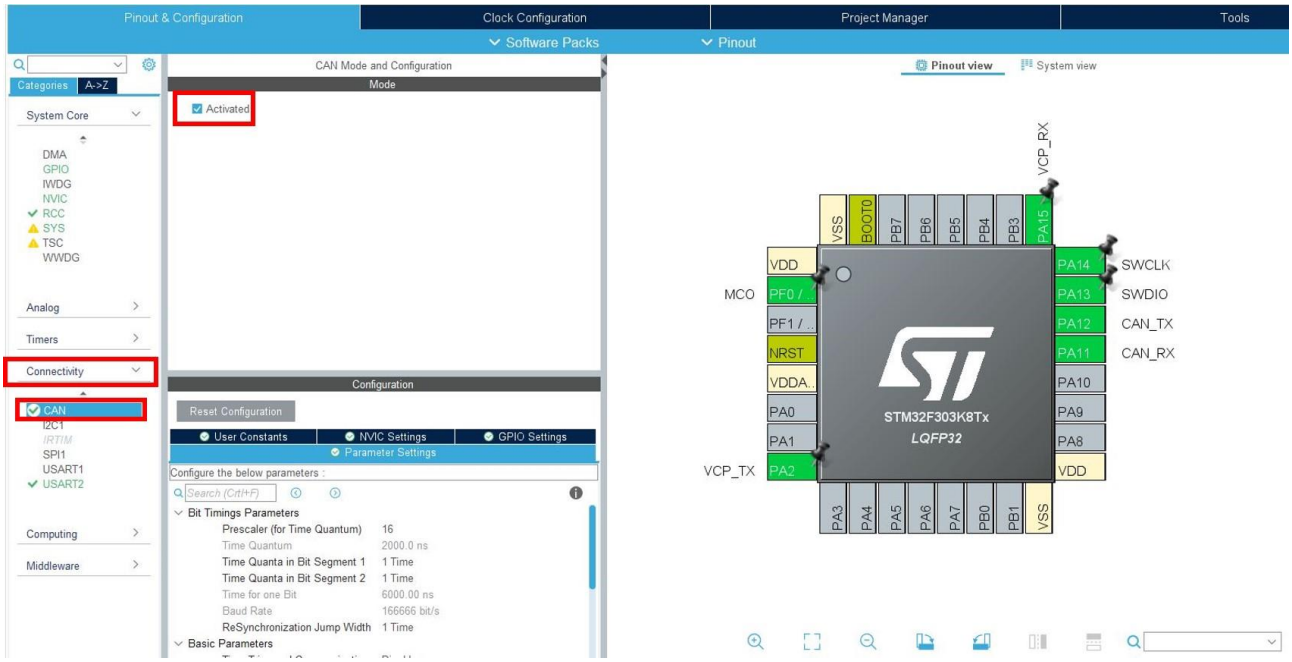
V. BUS CAN

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à l'émission et la réception de données via une liaison série de type CAN.
On supposera que le bus CAN été créé et est associée à l'instance *hcan1*.
On supposera également que le microcontrôleur STM32 est le maître (master) de la communication CAN et que les périphériques sont les esclaves (slave).

1. Initialiser un bus CAN

L'objectif de cette partie est de présenter la méthodologie à suivre pour initialiser une liaison série I2C sur le microcontrôleur STM32 à l'aide du logiciel STM32CubeMX.

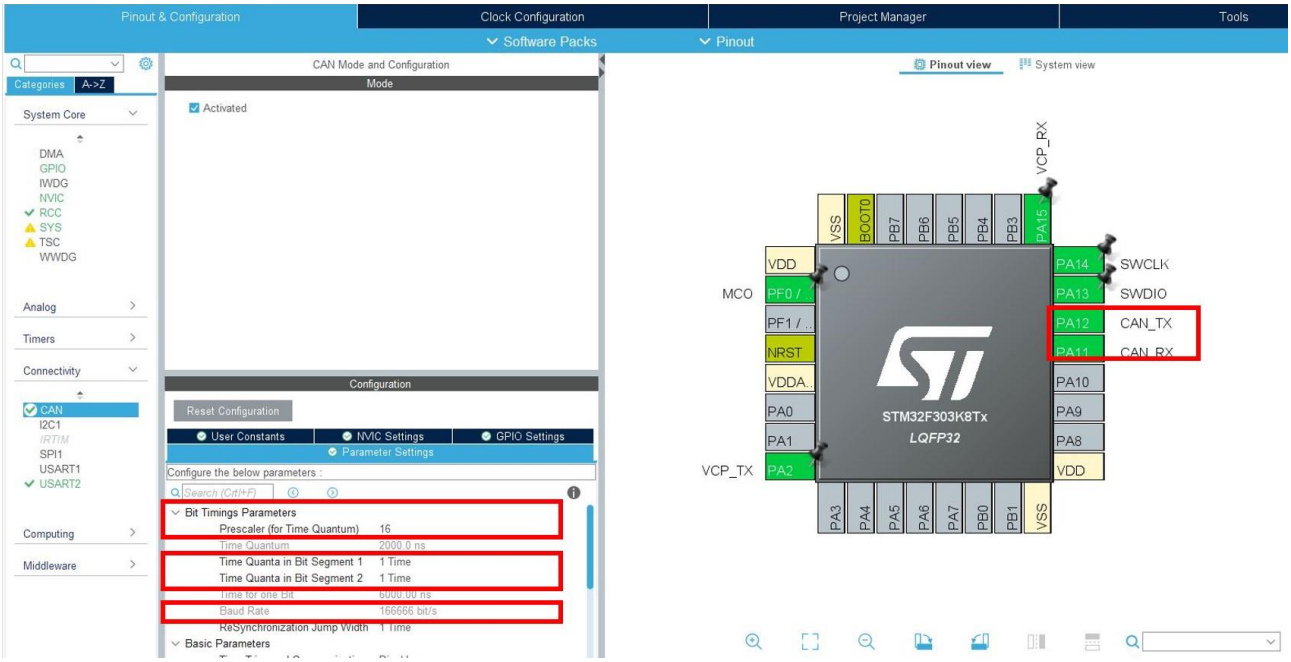
- Lancer le logiciel STM32CubeIDE
- Créer un nouveau projet lié à votre microcontrôleur STM32
- Dans l'onglet de gauche « Connectivity », sélectionner CAN puis cocher la case « Activated »



Les broches du microcontrôleur STM32 attribuées à ce bus CAN apparaissent.

- Les paramètres de configuration du bus CAN sont disponibles :
  - Prescaler
  - Time Quanta in Bit Segment 1
  - Time Quanta in Bit Segment 2

Ces 3 paramètres permettent de fixer le débit du bus CAN.



- Ajuster ces 3 paramètres de façon à obtenir le débit binaire souhaité

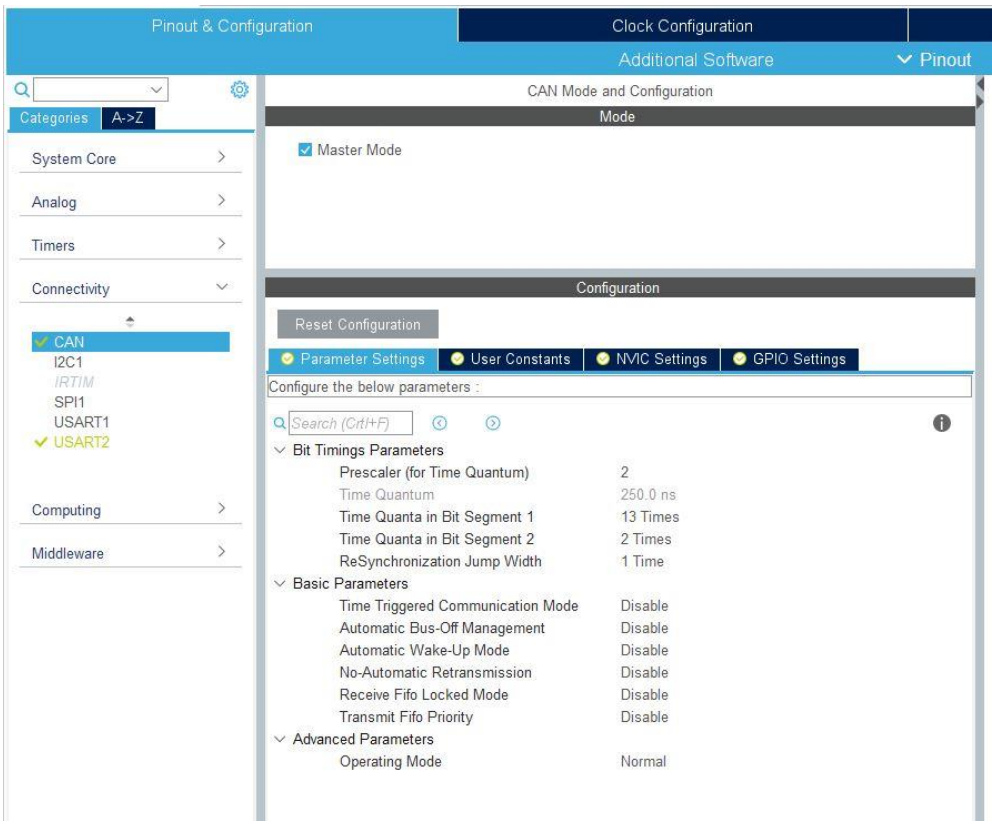
*Le débit binaire peut être calculé à partir de la durée d'un bit. La durée d'un bit est calculée par le formule suivante : Tbit = Tquantum * (Time Quanta Bit Segment 1 + Time Quanta Bit Segment 2 + Resynchronistation Jump Width).*

Pour vous aider à configurer ces paramètres, vous pouvez utiliser le site internet suivant :
http://www.bittiming.can-wiki.info/

2. Transmettre un message sur un bus CAN

L'objectif de cette partie est de présenter les instructions en langage C nécessaires à la transmission de données via un bus CAN.
On supposera que le bus CAN a été créé et est associé à l'instance *hcan*.
L'exemple de programme ci-dessous permet de transmettre une trame CAN. Le bus CAN est configuré de la manière suivante :

- mode étendu
- trame de donnée
- CAN ID = 0x446
- taille de donnée à transmettre= 8 octets
- donnée à émettre stockée dans la variable *CAN_TX*
- donnée à transmettre = [0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08]
- Dans STM32CubeMX, activer le mode « *Normal*» dans l'onglet «Parameters Settings => Advanced Parameters=> Operating Mode»



- Sauvegarder votre fichier d'initialisation

- Dans l'en-tête du programme principal *main.c*, déclarer les variables suivantes : TxHeader, TxData et TxMailBox.

```
/* USER CODE BEGIN PV */
CAN_TxHeaderTypeDef TxHeader; //Variable contenant la configuration du bus CAN
uint8_t TxData[8];            //Variable contenant la donnée sur 8 octets à trans-
metttre
uint8_t TxMailbox;            //Variable permettant la transmission des données

/* USER CODE END PV */
```

- Dans le programme principal *main.c*, configurer la trame CAN conformément au cahier des charges :

```
/* USER CODE BEGIN 2 */
TxHeader.DLC = 8 ;                      //Configuration du DLC de la trame CAN
TxHeader.IDE = CAN_ID_EXT ;             //Configuration de la trame CAN au format étendu
TxHeader.ExtId = 0x406 ;                //Configuration du CAN-ID de la trame CAN
TxHeader.RTR = CAN_RTR_DATA ;           // Configuration du bit RTR de la trame CAN
TxHeader.TransmitGlobalTime = DISABLE;     //Désactivation du Time Out Global

//Configuration des 8 octets de donnée à transmettre
TxData[0] = 0x01 ;
TxData[1] = 0x02 ;
TxData[2] = 0x03 ;
TxData[3] = 0x04 ;
TxData[4] = 0x05 ;
TxData[5] = 0x06 ;
TxData[6] = 0x07 ;
TxData[7] = 0x08 ;

HAL_CAN_Start(&hcan) ;          // Démarrage du bus CAN

/* USER CODE END 2 */
```

- Dans la boucle *while* du programme principal *main()*, ajouter les lignes de codes suivantes afin d'émettre la donnée sur le bus CAN configurée :

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
While (1)
{
    HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox) ;   // Emission de la
trame CAN
    HAL_Delay(500) ;

     /* USER CODE END WHILE */

}
```

Pour en savoir plus, consulter les vidéos suivantes :

- https://www.youtube.com/watch?v=JfWlIY0zAIc&t=436s:
- https://controllerstech.com/can-protocol-in-stm32/
- https://www.youtube.com/watch?v=KHNRftBa1Vc&list=PLfIJKC1ud8gjzwOPq9fvQt38Ut7EjsgwI&index=1
- http://wiki.labaixbidouille.com/index.php/Mise_en_place_d%27un_bus_CAN

3. Recevoir un message CAN

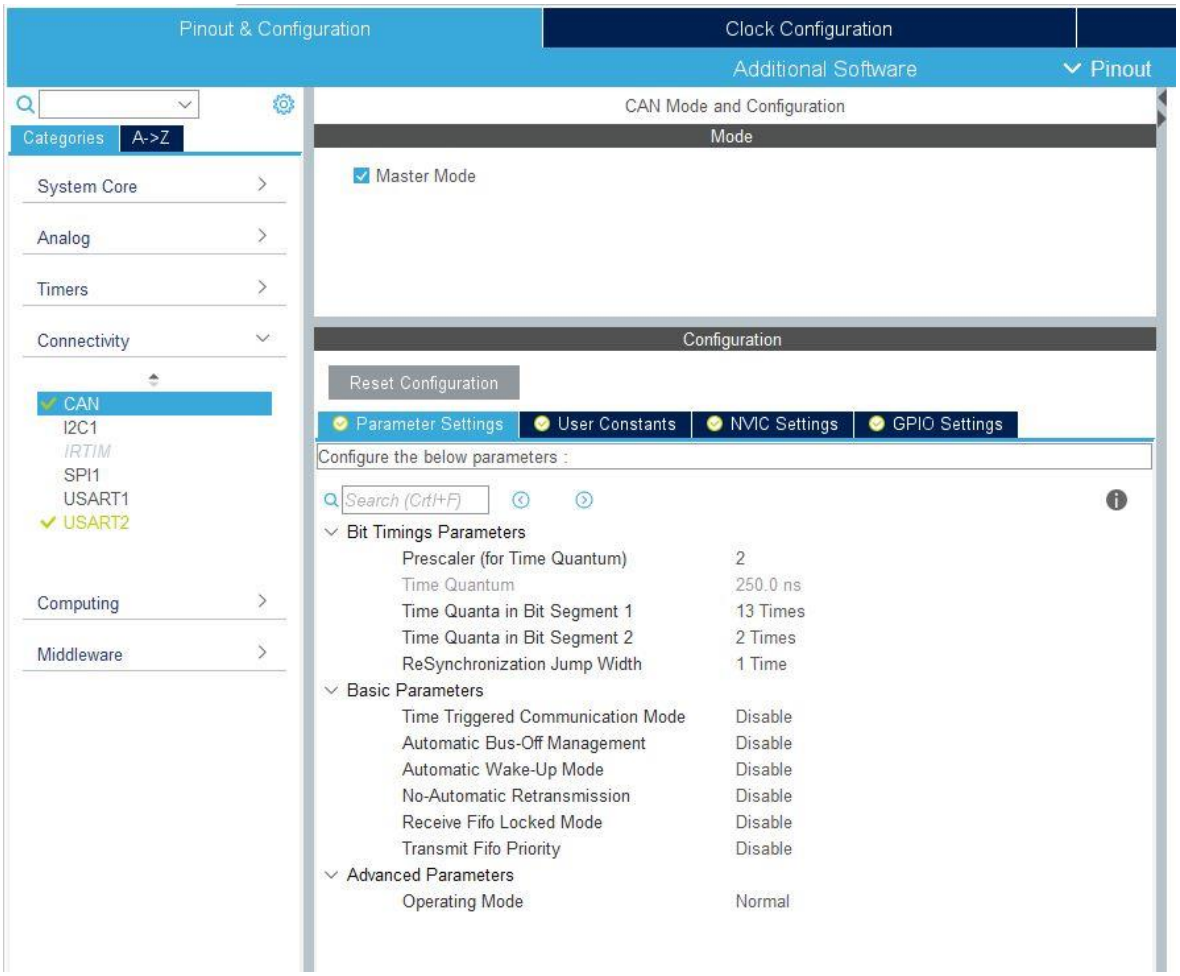
L'objectif de cette partie est de présenter les instructions en langage C nécessaires à la réception de données via un bus CAN.
On supposera que le bus CAN a été créé et est associé à l'instance *hcan*.
L'exemple de programme ci-dessous permet de recevoir une trame CAN. Les données sont reçues en mode interruption.
La trame CAN à recevoir est configurée de la manière :

- mode standard
- taille de donnée à recevoir = 6 octets
- CAN ID = 0x105 ;

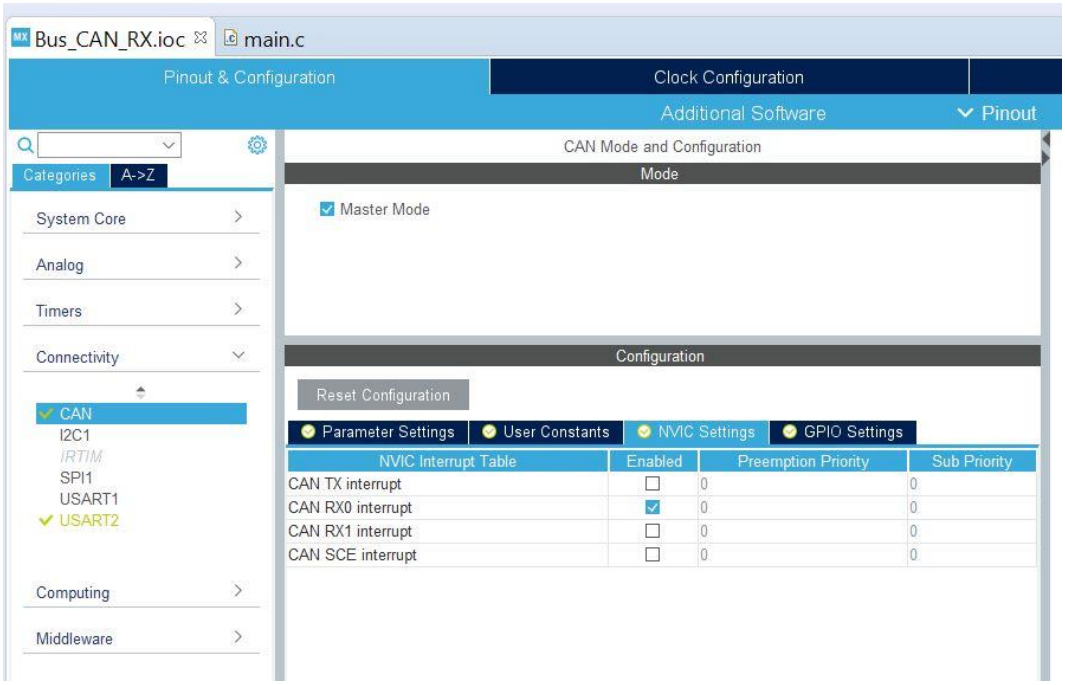
a. Configuration du bus CAN en réception

La procédure à suivre est la suivante :

- Configuration du bus CAN en mode normal :



- Activer les interruptions pour la réception de donnée CAN :



- Câbler le microcontrôleur à un adaptateur CAN

b. Programmation en C pour la réception d'une trame CAN

- Dans l'en-tête du fichier *main.c*, déclarer les variables suivantes : RxHeader, RxData:

```
/* USER CODE BEGIN PV */
CAN_RxHeaderTypeDef RxHeader; //Variable contenant la configuration du bus CAN en Rx
uint8_t RxData[6];            //Variable contenant la donnée sur 6 octets à recevoir

/* USER CODE END PV */
```

- Dans la fonction principale *main*, ajouter les lignes de codes suivantes afin de configurer le bus CAN comme définie dans le cahier des charges:

```
/* USER CODE BEGIN 2 */
HAL_CAN_Start(&hcan) ;          // Démarrage du bus CAN
CAN_FilterTypeDef canfilterconfig; //Variable pour configurer la réception CAN

//Configuration du filtre de réception CAN
canfilterconfig.FilterActivation = CAN_FILTER_ENABLE; // Activation du filtre de réception CAN
canfilterconfig.FilterBank = 10; // Bank de filtre utilisé
canfilterconfig.FilterFIFOAssignment = CAN_FILTER_FIFO0; // FIFO0 pour réception CAN
canfilterconfig.FilterIdHigh = 0x105<<5; // Config Filtre de réception CAN partie High
canfilterconfig.FilterIdLow = 0x0000; // Config Filtre de réception CAN partie Low
canfilterconfig.FilterMaskIdHigh = 0x105<<5; // Config Masque Filtre de réception CAN partie High
canfilterconfig.FilterMaskIdLow = 0x0000; // Config Masque Filtre de réception CAN partie Low
canfilterconfig.FilterMode= CAN_FILTERMODE_IDMASK; // Config Mode Filtre
canfilterconfig.FilterScale= CAN_FILTERSCALER_32BIT; // Config Echelle Filtre
canfilterconfig.SlaveStartFilterBank= 0; // Config Bank Filtre

//Activation de la réception CAN configurée précédemment
HAL_CAN_ConfigFilter(&hcan, &canfilterconfig);          // Configuration Filtre Rx
HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO0_MSG_PENDING); // Activation interruption Rx

/* USER CODE END 2 */
```

- Créer la fonction *HAL_CAN_RxFifo0MsgPendingCallback* associée à l'interruption en réception des données du bus CAN en ajoutant les lignes de codes suivantes :

```
/* USER CODE BEGIN 4 */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
	if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, CANRxData) != HAL_OK)
	{
		Error_Handler();
	}

	if (RxHeader.StdId == 0x105){
/// LIGNE DE CODE PERSONNELLE LORSQUE LA RECEPTION CAN EST OPERATIONNEL
	}

	else {

	}
}
/* USER CODE END 4 */
```

Remarque: Être bien rigoureux dans les lignes de code à ajouter.

c. Mise en œuvre finale d'une réception d'une trame CAN

- Relier le transmetteur de trame CAN à l'adaptateur CAN
- Compiler le programme STM32 en mode debugger
- Débugger le programme STM32
- Visualiser la variable *RX_Data* du STM32
- Configurer le transmetteur de trame CAN de façon à émettre une trame CAN avec les paramètres suivants :
  - Format standard
  - CAN-ID = 0x105
  - Taille des données = 6 octets
  - Données à définir par vous-même (6 octets à définir)
  - Débit binaire : conforme à la configuration du bus CAN sur le STM32
- Transmettre la trame CAN avec le transmetteur CAN
- Visualiser la donnée stockée dans la variable *RX_Data*

La variable dans *RX_Data* doit être la même que celle transmises par le transmetteur CAN.
Si vous modifiez les données dans la trame transmise par l'émetteur CAN, les données stockées dans *RX_Data* doit également changer.

4. Communication NMEA2000 avec l'afficheur B&G Vulcan7

L'objectif de cette partie est de présenter la communication à mettre en place entre un carte émettant des données au format NMEA2000 et l'afficheur B&G Vulcan7 utilisé dans le système d'aide à la navigation maritime disponible au lycée.

On supposera que l'émission de trames au format NMEA2000 via le bus CAN est opérationnel sur un microcontrôleur du type STM32. Si ce n'est pas le cas, se référer à la partie « Transmettre une donnée sur un bus CAN » de ce tutoriel.

La communication entre la carte et l'afficheur B&G doit permettre à l'afficheur d'afficher la donnée transmise par la carte sur son IHM.

Pour cela, la carte doit transmettre un ensemble de trame NMEA2000 afin qu'elle soit reconnue par l'afficheur B&G.

Voici l'ensemble de trame qui doivent être transmise par la carte :

Message à envoyer uniquement lors de la mise sous tension (initialisation):

Message à envoyer régulièrement :

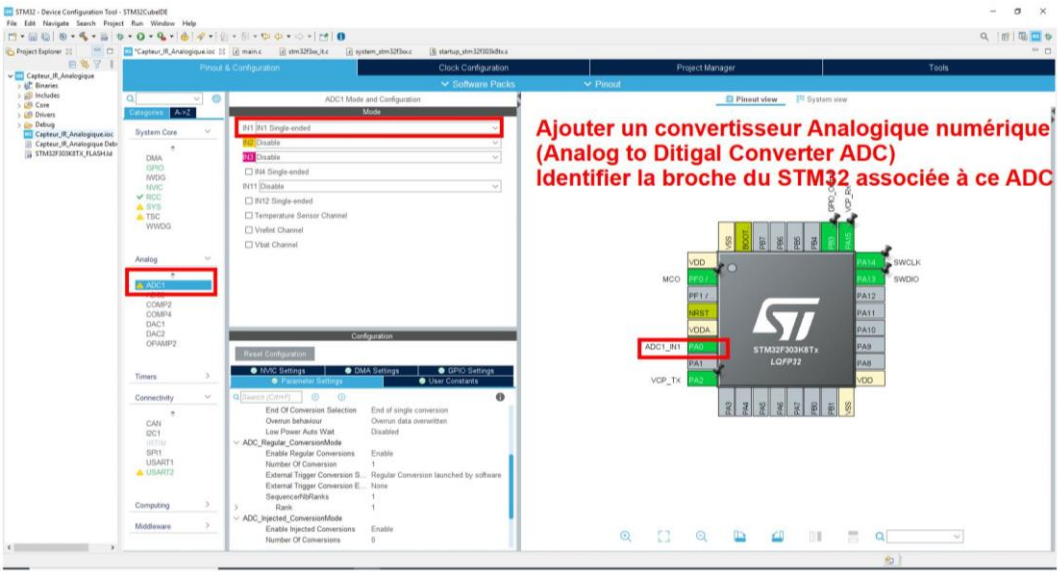
VI. CONVERSION ANALOGIQUE/NUMÉRIQUE

L'objectif de cette partie est de présenter la mise-en-œuvre d'un convertisseur analogique-numérique (CAN). En anglais, la conversion analogique-numérique se dit « Analog to Digital Converter » ou ADC.
La conversion analogique-numérique est utile dans le cadre de capteur analogique, capteur de distance infrarouge par exemple. Ce type de capteur génère une tension dont l'amplitude varie en fonction de la distance mesurée.

1. Initialisation du CAN

- Avec le logiciel STM32CubeMX, ajouter un CAN (ou ADC en anglais) à votre microcontrôleur et configurer le comme l'exemple ci-dessous

CAN



- Laisser tout par défaut et résoudre automatiquement le problème d'horloge (onglet *Clock Configuration*)
- Sauvegarder votre fichier et générer le code d'initialisation.
- Sur votre platine de prototypage, relier le capteur analogique à la broche d'entrée de votre Convertisseur Analogique Numérique et alimenter le capteur.

2. Programmation en C

Le programme en C consiste à démarrer le CAN, effectuer la conversion en mode Polling (plus de détail sur internet) et à stocker la valeur numérique dans une variable. Voici les lignes de code à ajouter dans le fichier *main.c* :

- Dans le fichier *main.c*, créer une variable de type *int* appelée ADC_RES
- Dans le fichier *main.c* dans la fonction *main* et la boucle *while*, ajouter le code suivant :

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
int ADC_RES                    // Valeur de la conversion analogique => numérique

while (1)
{
      // Start ADC Conversion
      HAL_ADC_Start(&hadc1);

      // Poll ADC1 Peripheral & TimeOur = 1ms
      HAL_ADC_PollForConversion(&hadc1, 1);

      // Read the ADC Conversion Result
      AD_RES = HAL_GetValue(&hadc1);
  /* USER CODE END WHILE */

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

Attention : les instructions *HAL_ADC_Start* et *HAL_ADC_PollForConversion* sont absolument à mettre avant l'instruction *HAL_ADC_GetValue*.

- Lancer la compilation et le débuggage et visualiser la variable *ADC_RES* afin de vérifier le bon fonctionnement de votre code

VII. TRAITEMENT DE DONNÉES

Ce chapitre présente des exemples de traitement de données. Avant cela, le chapitre présente les différents types de variable utilisés en langage C pour le STM32 ainsi que des exemples d'initialisation et d'opérateur logique.

1. Type de variables

Voici les différents types de variables utilisés en langage C :

Type de variable	Caractéristique
Entier	Nombre entier négatif ou positif
Chiffre à virgule	Chiffre à virgule négatif ou positif
Entier non-signé codé sur 8 bits	Nombre entier codé sur 8 bits donc compris entre 0 et 255 (255 = 2^8-1) A utiliser pour coder un caractère ASCII
Entier non-signé codé sur 16 bits	Nombre entier codé sur 16 bits compris entre 0 et 65 535 (65 535 = 2^{16}-1)
Entier non-signé codé sur 32 bits	Nombre entier codé sur 32 bits compris entre 0 et 4 292 967 295 (4 292 967 295 = 2^{32}-1)
Tableau de caractère ASCII	Tableau de plusieurs caractères ASCII A utiliser pour transmettre une trame comportant plusieurs caractères ASCII
Caractère non-signé	1 caractère non-signé codé sur 16 bits

2. Liste des opérateurs logiques

Voici la liste des opérateurs logiques disponibles en langage C

Opérateur logique	Opérateur en language C	Exemple de code
NON (NOT)	~	a = ~b;
ET (AND)	&	c = a & b;
OU (OR)	|	c = a | b;
OU EXCLUSIF (XOR)	^	c = a ^ b;
Décalage à droite (Right shift)	>>	b = a >>2; Décalage de 2 bits vers la droite
Décalage à gauche (Left shift)	<<	b = a <<3; Décalage de 3 bits vers la gauche

3. Conversion d'une chaîne de caractère ASCII vers un nombre entier

L'objectif de cette partie est de présenter la conversion d'une chaîne de caractère ASCII en un nombre entier.
La chaîne de caractère ASCII peut provenir d'une liaison série, I2C ou CAN associée à un capteur ou un périphérique.

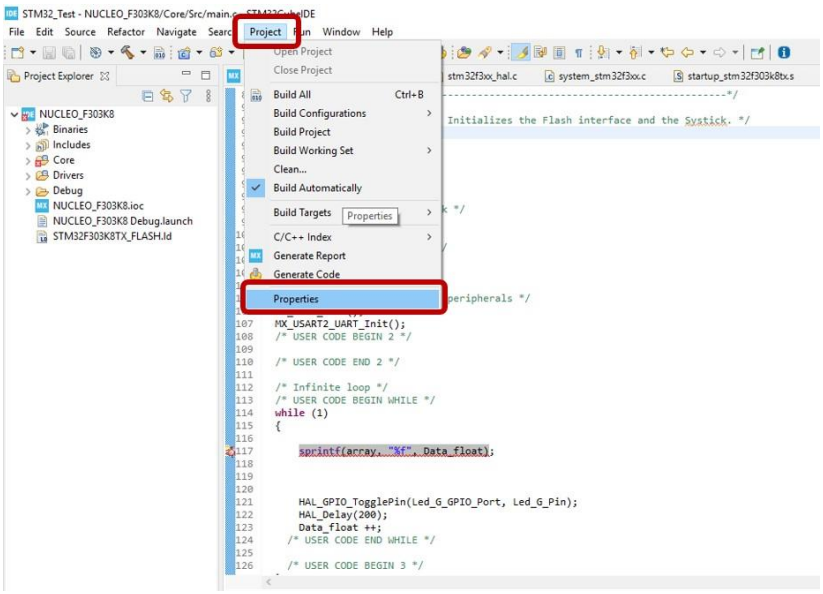
- Exemple de code pour convertir la chaîne de caractère ASCII « 4528 » en l'entier 4 528 :

```
  /* USER CODE BEGIN WHILE */
  char Data_char[]="4528";      // Variable de type chaîne de caractère à convertir

  //Création des variables de type int
  int Data_int_1, Data_int_2, Data_int_3, Data_int_4, Data_int_final;

  Data_int_1 = Data_Char[0] - 48;      // Conversion du premier caractère ASCII en entier
  Data_int_2 = Data_Char[1] - 48;      // Conversion du deuxième caractère ASCII en entier
  Data_int_3 = Data_Char[2] - 48;      // Conversion du troisième caractère ASCII en entier
  Data_int_4 = Data_Char[3] - 48;      // Conversion du quatrième caractère ASCII en entier

  // Calcul final
  Data_int_final = Data_int_1 * 1000 + Data_int_1 * 1000 + Data_int_2 * 100 + Data_int_3 *
10 + Data_int_4;
```

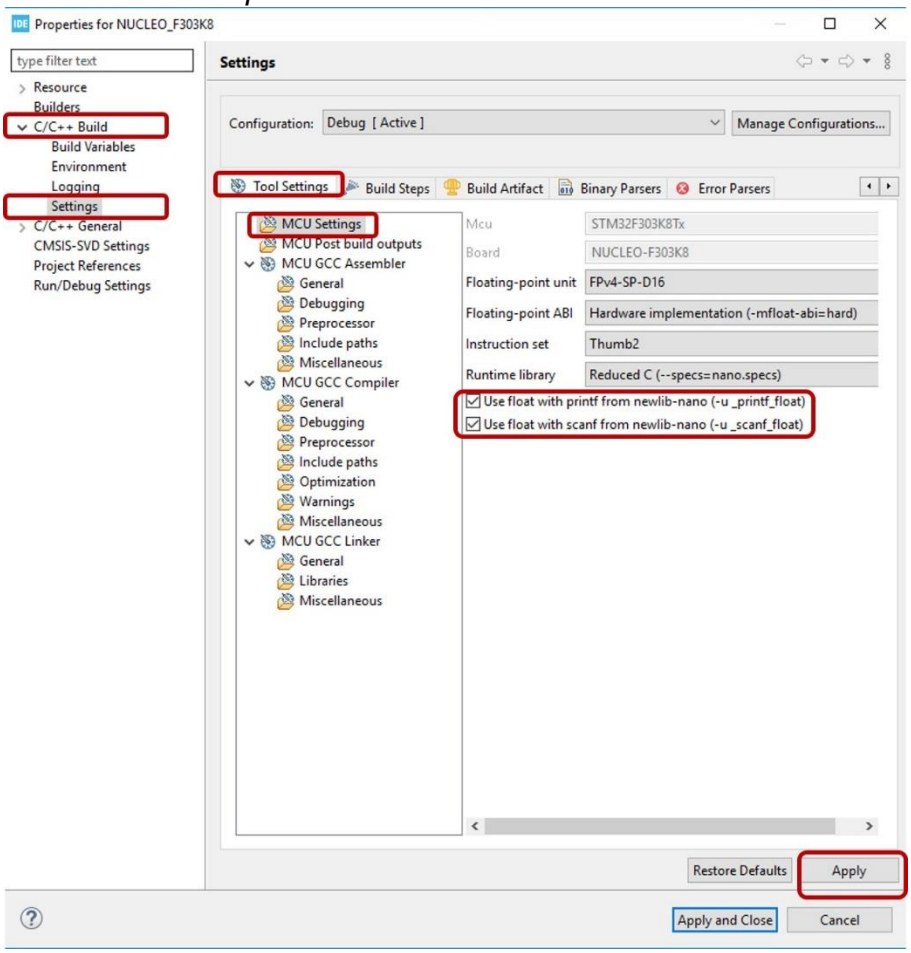
4. Conversion d'un nombre entier vers une chaîne de caractère ASCII

L'objectif de cette partie est de présenter la conversion d'un nombre entier en une chaîne de caractère ASCII.
Au préalable, le projet du logiciel STM32CUBEIDE doit être configuré de la manière suivante :

- Cliquer sur *Project => Properties*



- Dans la fenêtre qui apparait, aller dans C/C++ Build => Settings => Tool Settings => MCU Settings et cocher les 2 cases suivantes :
  - Use float with printf from newlib-nano (-u_printf_float)
  - Use float with scanf from newlib-nano (-u_scanf_float)



- Instruction à utiliser : **sprintf(char*, "%d", int) ;**
- Arguments de cette fonction :
  - **Char*** : Variable de type chaine de caractère à destination de la conversion
  - **%d** : conversion d'un nombre entier
  - **int** : Variable de type entier à convertir

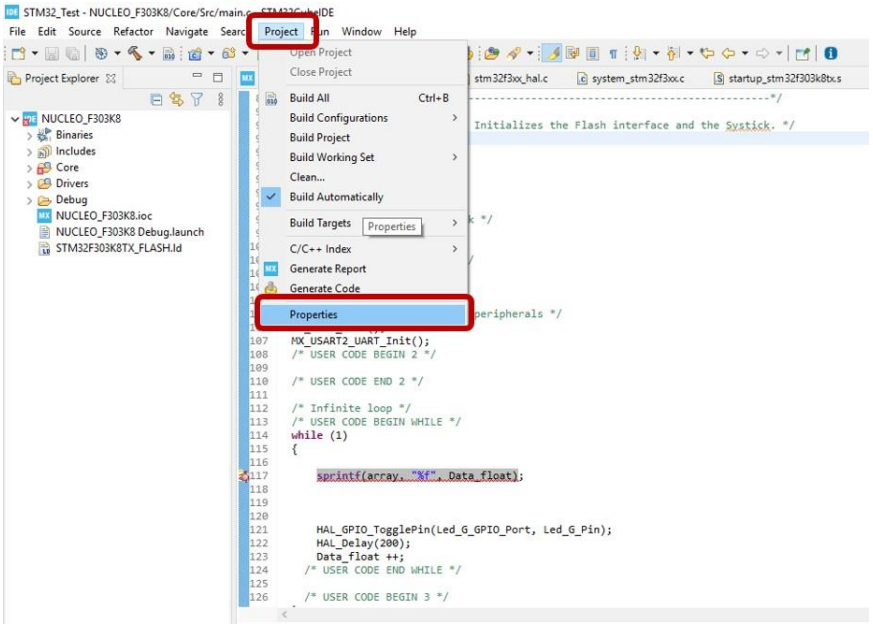
- Exemple de code pour convertir l'entier 4 523 en la chaîne de caractère ASCII « 4523 ».

```
/* USER CODE BEGIN WHILE */

//Création de variables
int Data_int = 4523;          // Variable de type int à convertir
uint8_t Data_Tableau[4];      // Variable de tpe chaine de caractère

sprintf(Data_Tableau, "%d", Data_int); // Conversion
```

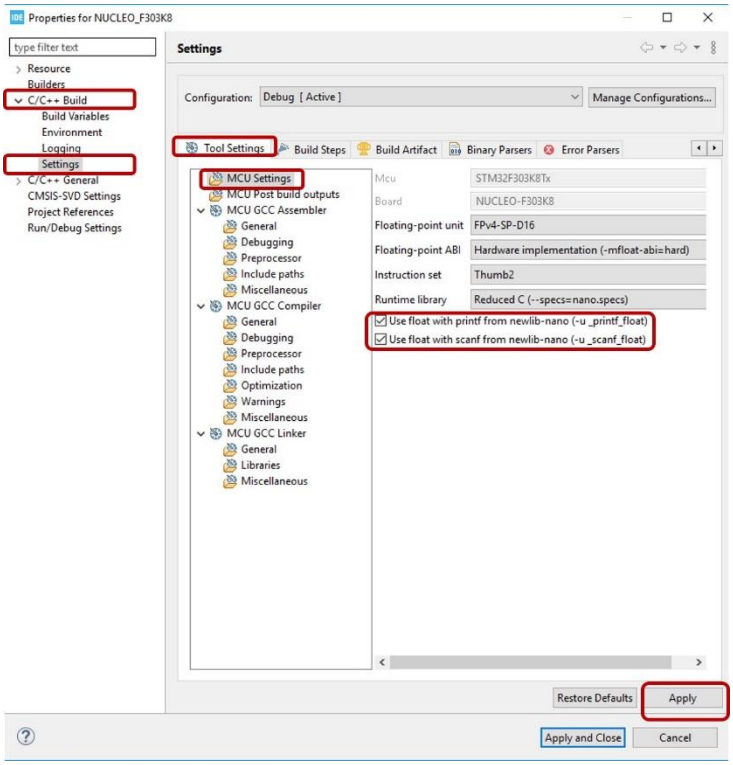
5. Conversion d'un nombre à virgule (type *float*) vers une chaîne de caractère

L'objectif de cette partie est de présenter la conversion d'un nombre à virgule en une chaîne de caractère ASCII.
Au préalable, le projet du logiciel STM32CUBEIDE doit être configuré de la manière suivante :

- Cliquer sur *Project => Properties*



- Dans la fenêtre qui apparait, aller dans *C/C++ Build => Settings => Tool Settings => MCU Settings* et cocher les 2 cases suivantes :
  - *Use float with printf from newlib-nano (-u_printf_float)*
  - *Use float with scanf from newlib-nano (-u_scanf_float)*



- Instruction à utiliser : **sprintf(char*, "%f", float) ;**
- Arguments de cette fonction :
  - **Char*** : Variable de type chaine de caractère à destination de la conversion
  - **%f** : conversion d'un nombre à virgule
  - **float** : Variable de type nombre à virgule à convertir

- Exemple de code pour convertir du nombre à virgule 35.9 en la chaîne de caractère ASCII « 35.9 ».

```
/* USER CODE BEGIN WHILE */

//Création de variables
float Data = 35.9;            // Variable de type float à convertir
uint8_t Data_Tableau[]; // Variable de tpe chaine de caractère

sprintf(Data_Tableau, "%f", Data); // Conversion
```

6. Conversion d'une chaîne de caractère ASCII vers un nombre à virgule

L'objectif de cette partie est de présenter la conversion d'une chaîne de caractère ASCII en nombre à virgule.

- Ajouter la bibliothèque <stdio.h> en début du programme :

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "i2c.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
/* USER CODE END Includes */
```

- Exemple de code pour convertir la chaîne de caractère ASCII « 12.6 » en un nombre à virgule 12.6 :

```
/* USER CODE BEGIN WHILE */

//Création de variables
uint8_t Data_Tableau[4]="12.6"; // Variable de type chaine de caractère à convertir
float Data;                     // Variable de type float

Data = atof(Data_Tableau);
```

7. Conversion d'un entier en décimal en hexadécimal dans une chaîne de caractère

```
//Création de variables
int Data_Entier=1234;     // Variable de type entier en décimal à convertir
uint8_t Data_Decimal[2];  // Variable de type chaine de caractère en hexadécmila

sprintf(Data_Decimal, "%x", Data_Entier); // Conversion en [0x04][0xD2]
```

8. Concaténer plusieurs chaînes de caractère

L'objectif de cette partie est de présenter la concaténation de plusieurs chaînes de caractère.

Pour cela, 2 instructions sont utiles :

- Instruction pour copier une chaîne de caractère : **strcpy(char*, char*) ;**
- Arguments de cette fonction :
  - **char*** : Variable de type chaine de caractère de destination
  - **char*:** Variable de type chaine de caractère de source

- Instruction pour concaténer une chaîne de caractère : **strcat(char*, char*) ;**
- Arguments de cette fonction :
  - **char*** : Variable de type chaine de caractère de destination
  - **char*:** Variable de type chaine de caractère de source

9. Calcul d'un checksum basé sur le OU exclusif (XOR)

Très souvent, le checksum est basé sur le calcul d'un OU exclusif bit à bit (XOR) à partir des données. En langage C, l'opérateur OU exclusif bit à bit (XOR) est codé par le symbole ^

Par exemple pour calculer le checksum d'un tableau de donnée de type entier (int)
tab = {0,2,1,5,3} ;
Il faut coder : checksum = tab[0] ^ tab[1] ^ tab[2] ^ tab[3] ^ tab[4] ;

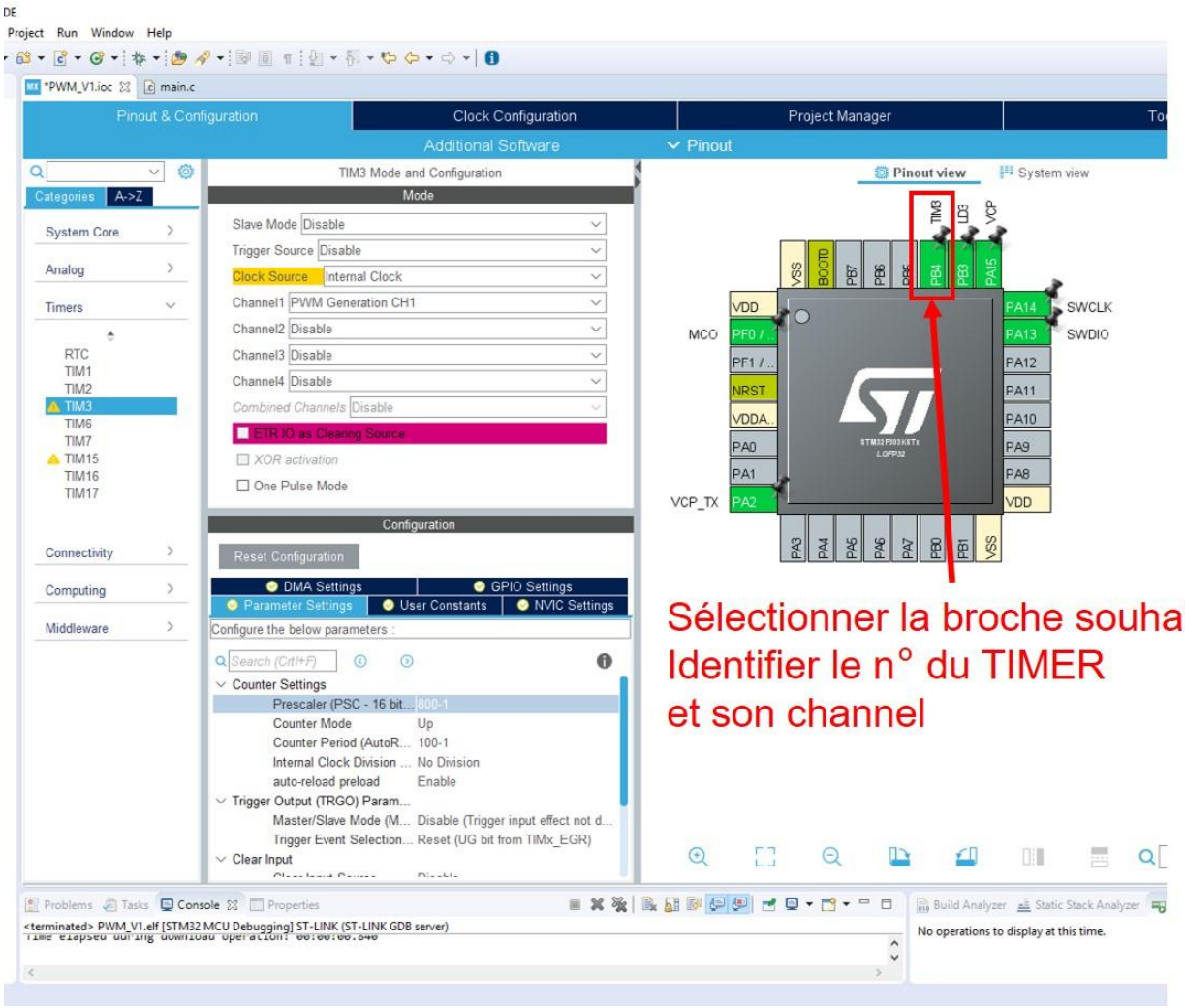
VIII. TIMER

Dans les microcontrôleurs, un timer est un système permettant de quantifier le temps réel. Pour cela, le microcontrôleur utilise une référence temporelle, horloge, dont la durée est fixe et connue puis il compte le nombre de cycle de cette horloge pour connaître le temps écoulé.
Les timer sont utilisés pour générer un signal modulé en largeur d'impulsion (Pulse Width Modulation ou PWM) ou pour distinguer un appui long d'un appui court sur un bouton-poussoir.

1. Signal PWM (Pulse Width Modulation)

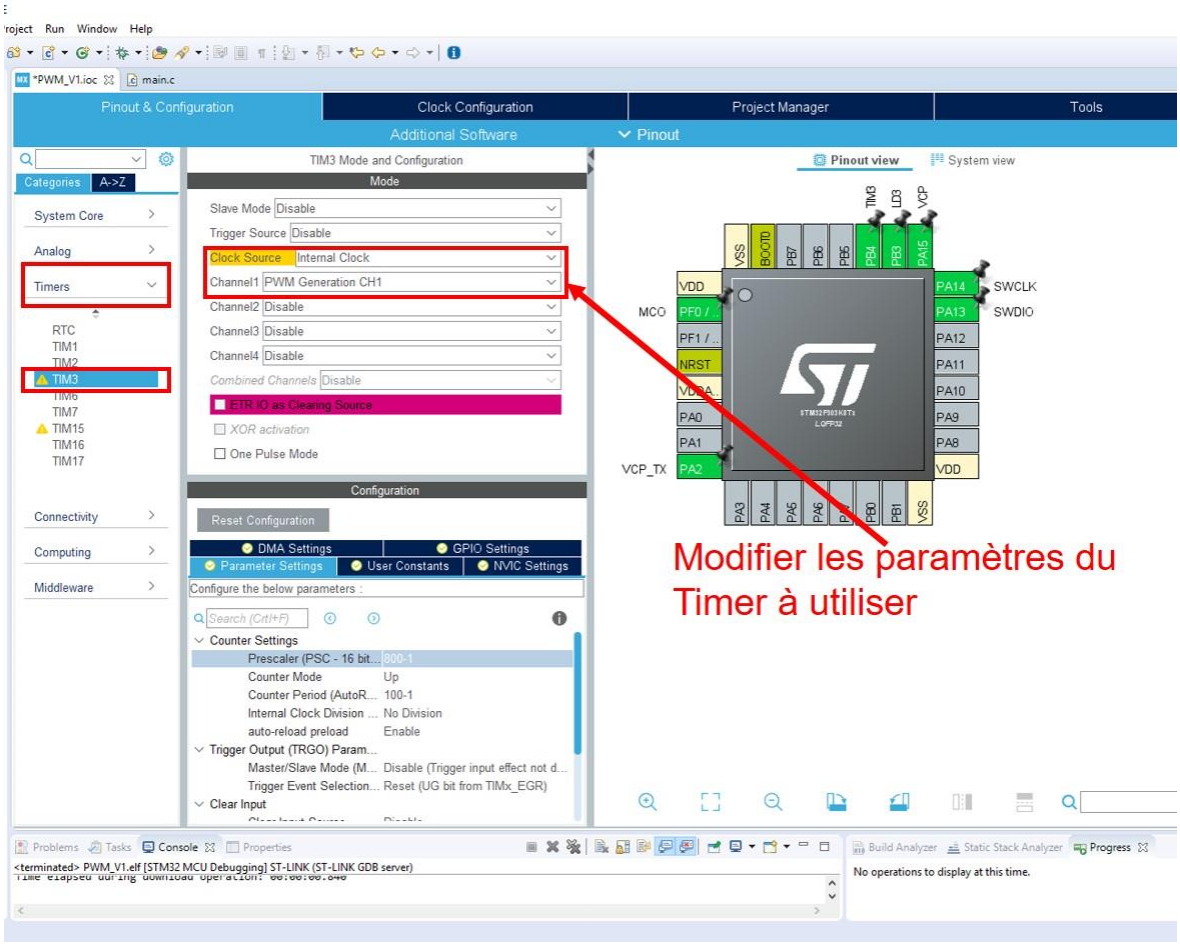
L'objectif de cette partie est de générer un signal PWM afin de commander une Led et d'en moduler l'intensité lumineuse.

- Sélectionner la broche du microcontrôleur STM32 et identifier le n° du timer et son channel

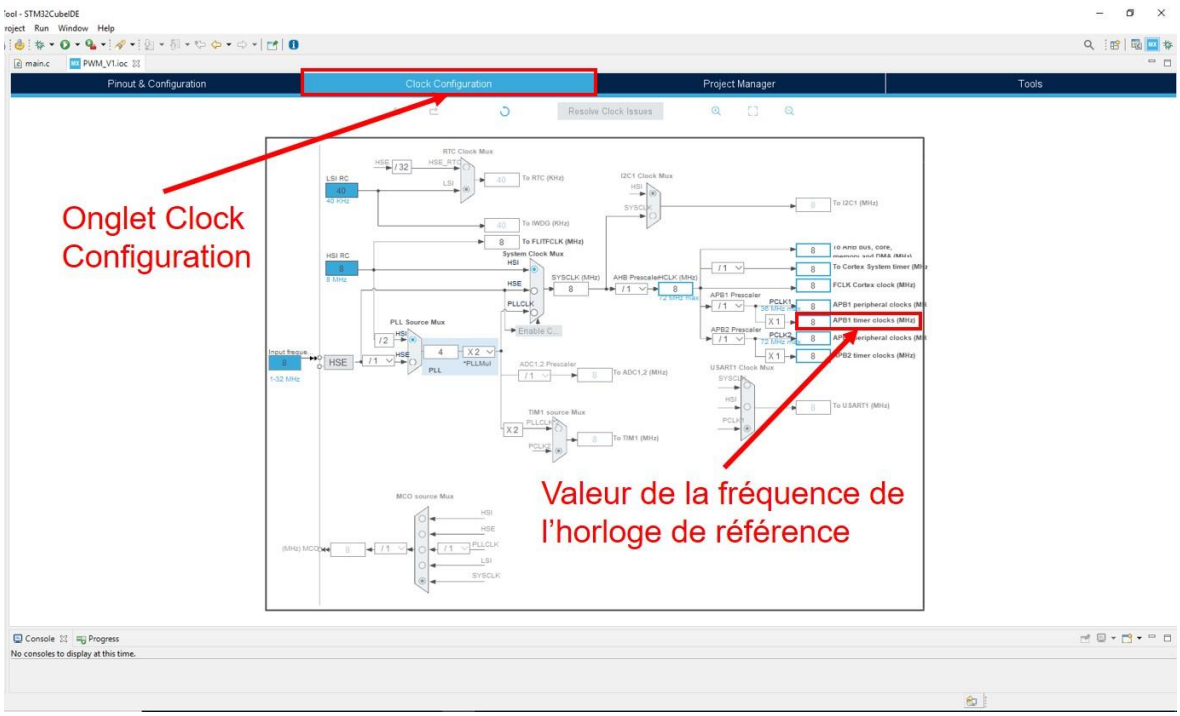


Dans cet exemple ci-dessus, la broche PB4 du microcontrôleur STM32F303K8 est associé au Timer 3 canal 1.

- Configurer le timer en sélectionnant l'horloge interne comme source et activer la génération du signal PWM

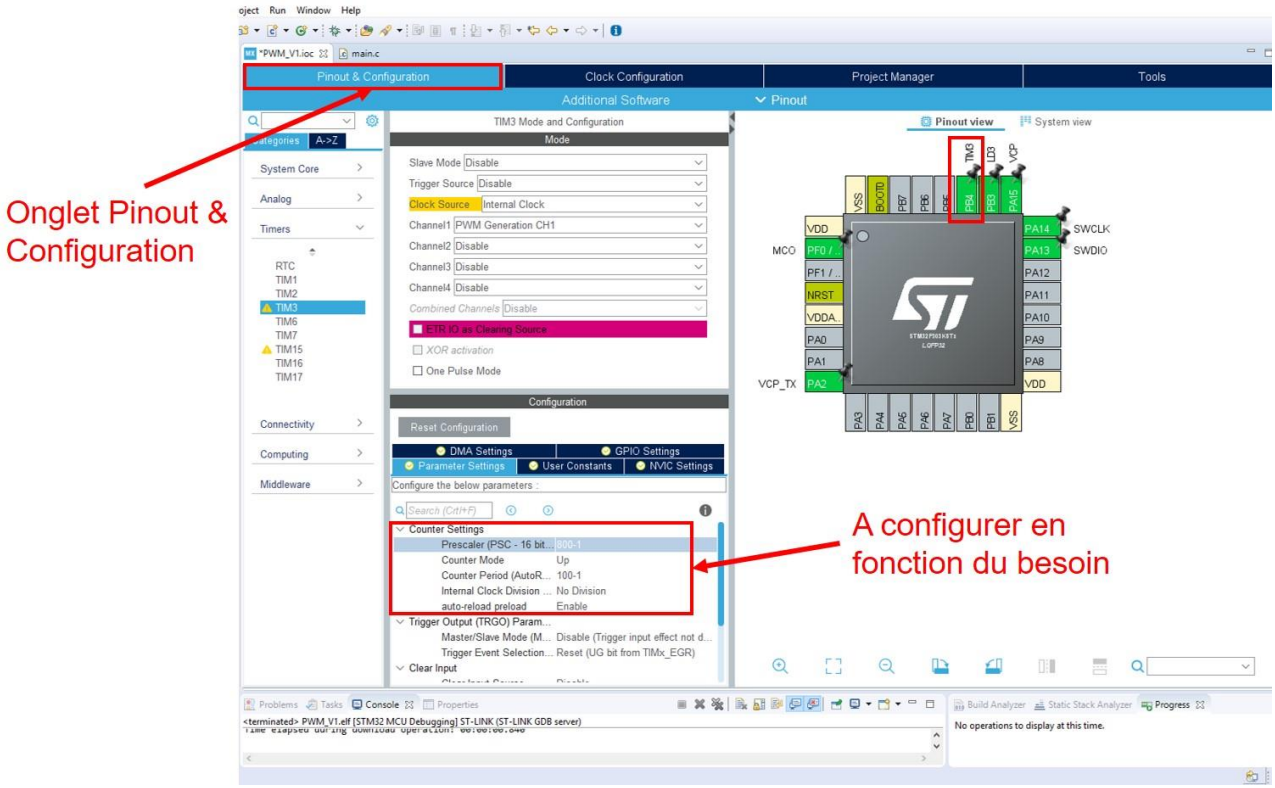


- Aller dans l'onglet *Clock Configuration* et identifier la fréquence de l'horloge de référence.



Dans cet exemple ci-dessus, la fréquence de l'horloge de référence du timer est fixée à 8 MHz. Il s'écoule donc 0.125µs à chaque cycle d'horloge.

- Retourner dans l'onglet et configurer les paramètres du timer : *Prescaler* et *Count Period* et *auto-reload preload* comme indiqué sur l'exemple ci-dessous.



L'*auto-reload preload* est activé (Enable) afin de recharger le compteur une fois qu'il a atteint son maximum.

Le *Prescaler (PSC)* et le *Counter Period (ARR)* permettent de définir la fréquence du signal PWM. La fréquence du signal PWM se calcule avec la formule suivante :

$$\mathrm{F}_{PWM} = \frac{F_{CLK}}{(ARR + 1) * (PSC + 1)}$$

Le *ARR* et le *CCR* permettent de définir le rapport cyclique de la PWM. Le rapport cyclique de la PWM (DT_{PWM}) se calcule avec la formule suivante :

$$\mathrm{DT}_{PWM} = \frac{CCR}{ARR + 1}$$

Le *PSC* et le *ARR* se définissent sur l'interface STM32CubeMX. Ils ne peuvent plus être modifier dans le programme. Le *CCR* se définit dans le code. Il peut être modifier au cours du programme.

Si l'on souhaite fixer la fréquence de la PWM à 50 Hz avec un rapport cyclique compris entre 0 et 100%, il faut fixer *PSC* à 1 599 et *ARR* à 99 car :

$$\mathrm{F}_{PWM} = \frac{F_{CLK}}{(ARR + 1) * (PSC + 1)} = \frac{8\,MHz}{(1599 + 1) * (99 + 1)} = \frac{8\,MHz}{(1600) * (100)} = 50\,Hz$$

$$\mathrm{DT}_{PWM} = \frac{CCR}{ARR + 1} = \frac{100}{99 + 1} = \frac{100}{100} = 1$$

Le CCR sera compris entre 0 et 100 dans le code.

- Générer le code d'initialisation
- Dans le fichier *main.c*, voici les instructions suivantes à utiliser :
  - **HAL_TIM_PWM_Start (*htim, channel);** *// démarrer la PWM*
  - **htim->CCR1 = … ;** *// Configuration du rapport cyclique (Duty Cycle de la PWM)*

- Instruction à utiliser : **HAL_TIM_PWM_Start (*htim, channel);**
- Arguments de cette fonction :
  - ***htim** : pointeur vers l'instance du timer à utiliser
  - **channel**: nom du channel à utiliser

- Instruction à utiliser : **htim->CCR1 = …. ;**
- Arguments de cette fonction :
  - ***htim** : pointeur vers l'instance du timer à utiliser
  - **CCR1**: valeur du rapport cyclique

Voici un exemple de code pour générer un signal PWM associé au timer 3 et au canal1.

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);  // Démarrage du timer 3 canal 1 pour la PWM
  while (1)
  {
        for (int i=0; i<11; i ++)
        {
              TIM3->CCR1 = i*10;// Rapport cyclique = i*10% avec i compris entre 0 et 10
              HAL_Delay(100);
        }


    /* USER CODE END WHILE */

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

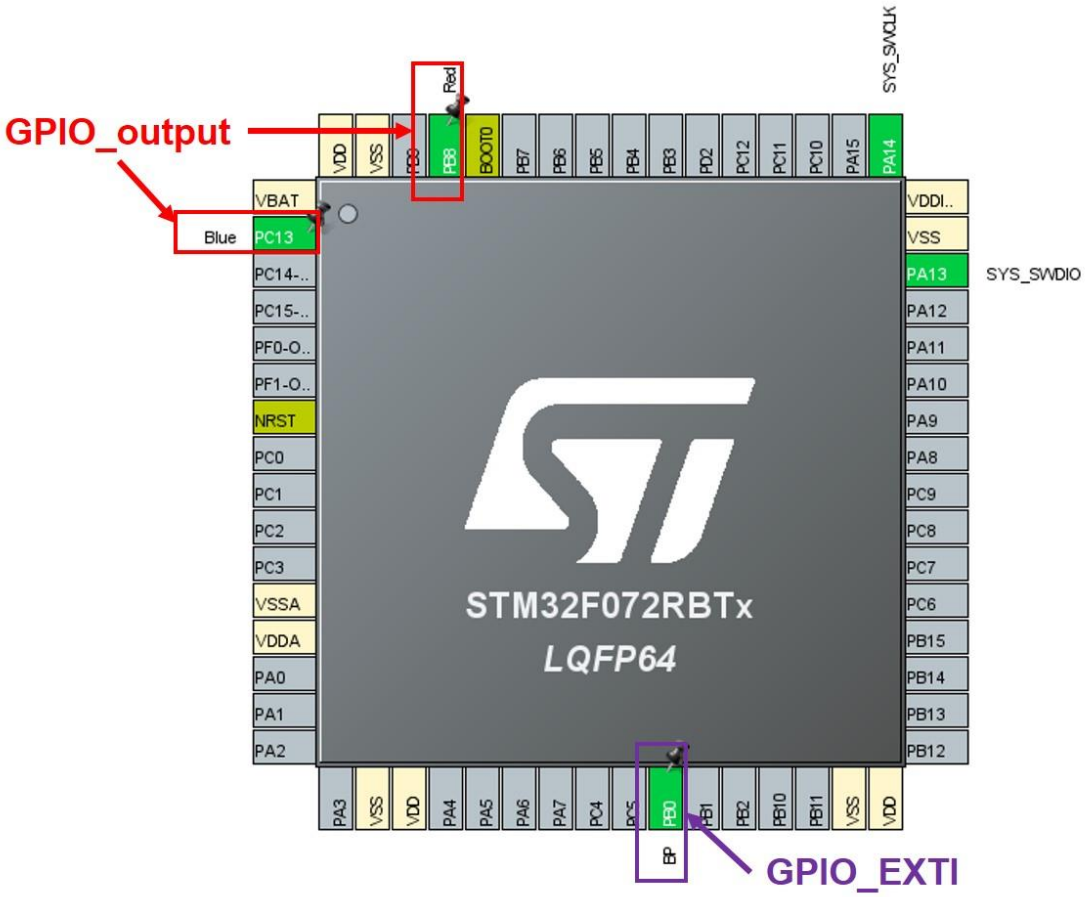
2. Distinction d'un appui long – appui court sur un bouton-poussoir

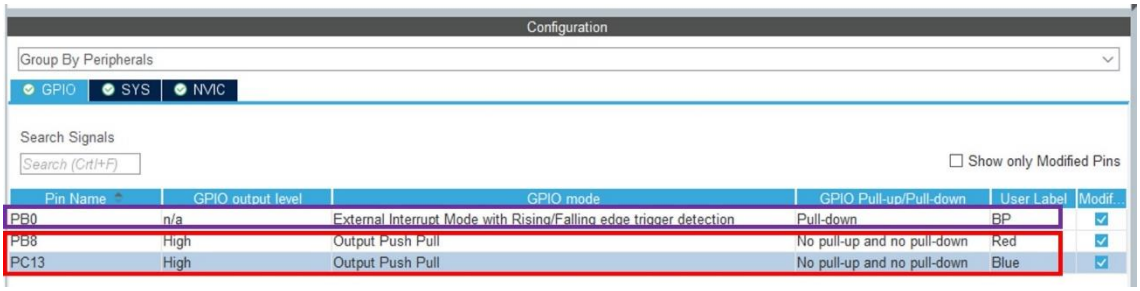
L'objectif de cette partie est de présenter la méthodologie à suivre pour distinguer un appui long d'un appui cours sur un bouton-poussoir.

Cet exemple est présenté sur un microcontrôleur STM32F072RBT6, mais la méthodologie est la même pour un autre type de microcontrôleur de la famille STM32.
Dans cet exemple, l'objectif est soit d'allumer en bleue une led RGB s'il y a un appui court sur le bouton-poussoir, soit d'allumer en rouge une led RGB s'il y a un appui long sur le bouton-poussoir.
Le bouton-poussoir est relié à la broche PB0 du microcontrôleur.
La led RGB est reliée aux broches PB8 pour la led rouge et PC0 pour la led bleue. Petite subtilité, la led RGB est à anode commune. C'est-à-dire qu'il faut mettre la broche du microcontrôleur à l'état bas pour allumer la led souhaitée.
Pour faire cela, il sera nécessaire d'activer une interruption sur le bouton-poussoir et d'utiliser un *timer* pour compter le temps entre l'appui et le relâchement du bouton-poussoir.

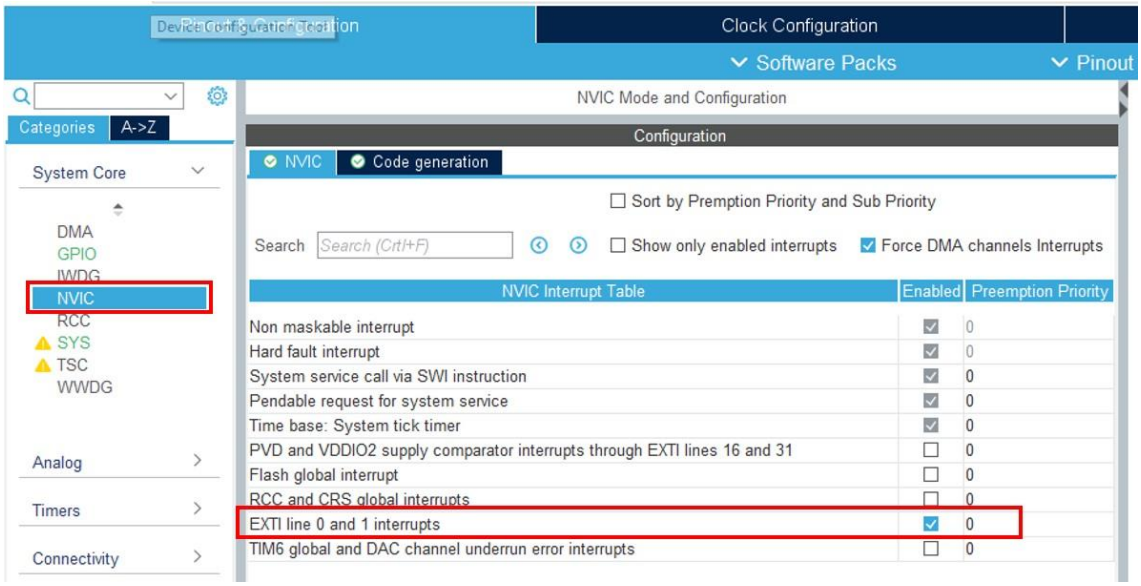
- Sur l'interface CubeMX, initialiser les broches du microcontrôleur

Les broches PB8 et PC0 : *GPIO_output* avec un pull-down et un état de référence à l'état haut (Led éteinte)
La broche PB0 : *GPIO_EXTI* (activation de l'interruption) en mode « *External Interrupt Mode with Rising/Falling edge trigger detection* ».

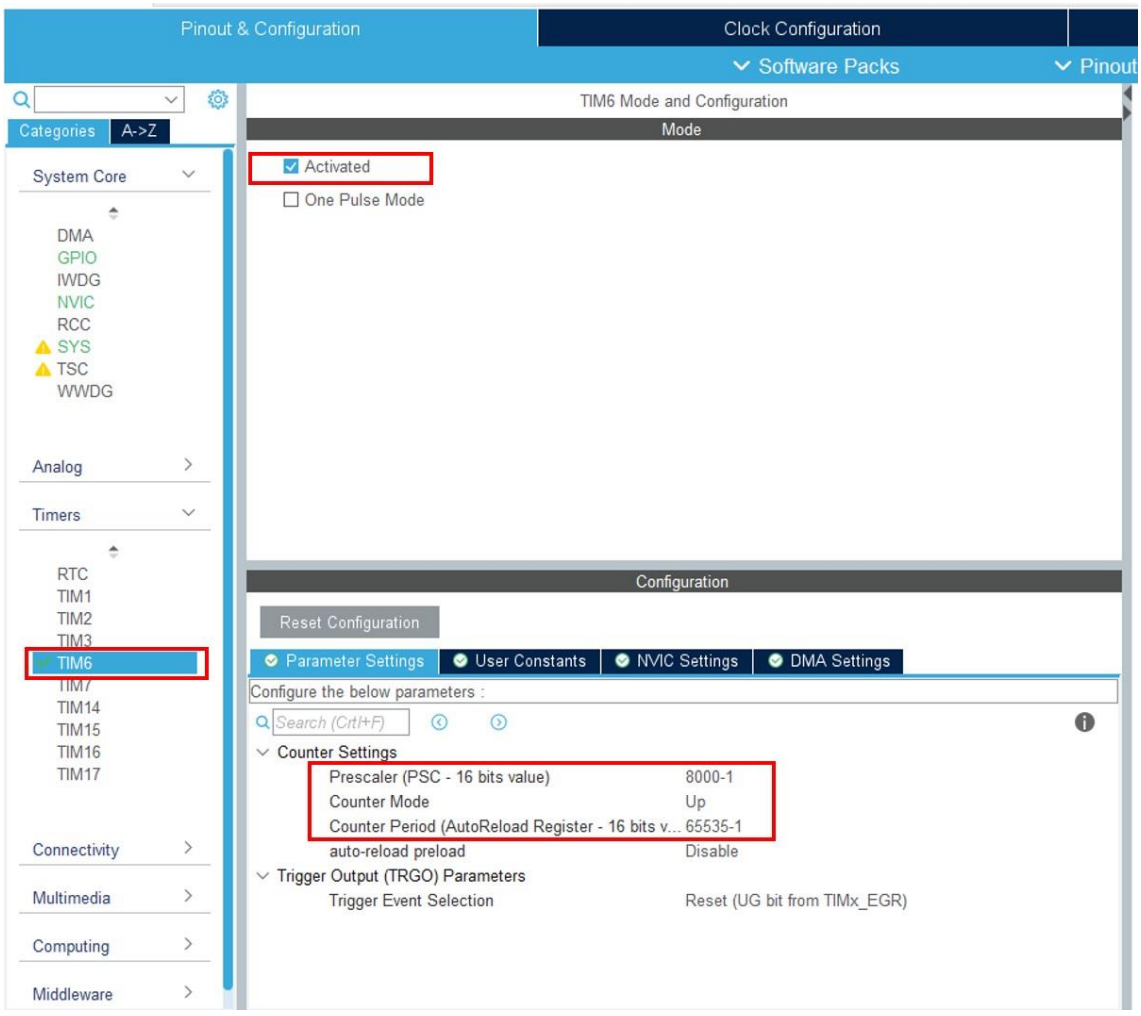




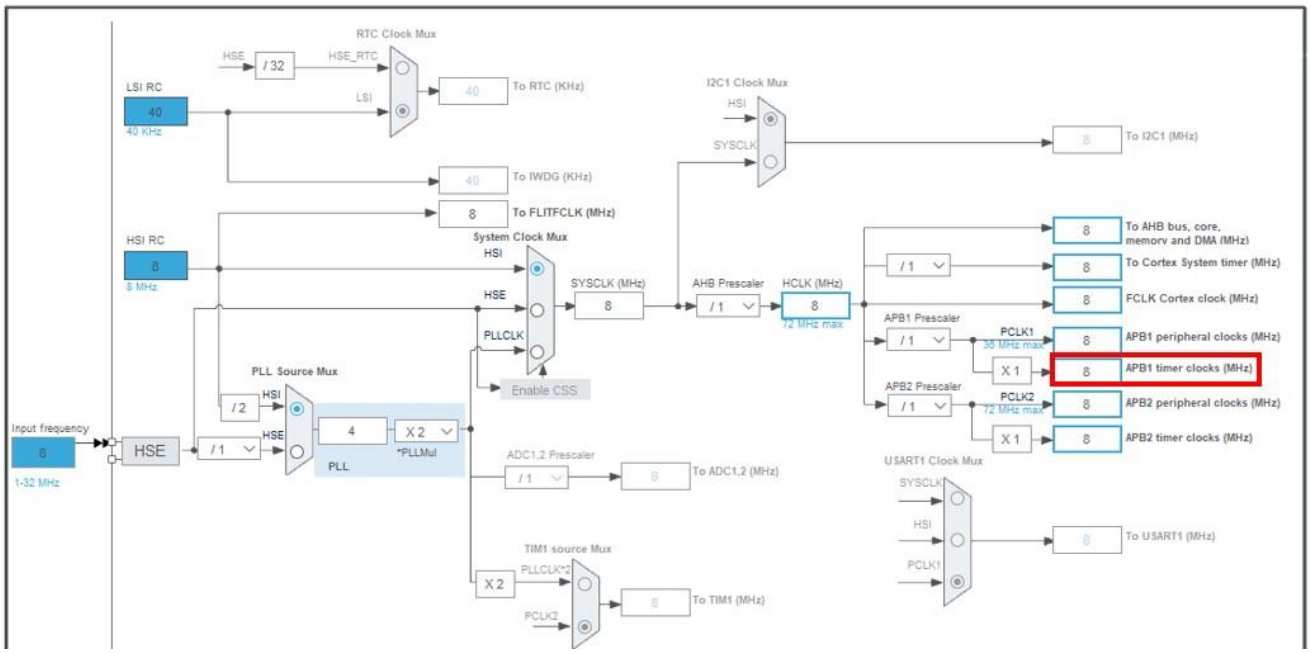
- Sur l'interface CubeMX, activer les interruptions en allant dans l'onglet « *NVIC* » et en cochant la case « *EXTI line 0 and 1 interrupt* ».



- Sur l'interface CubeMX, activer le timer TIM 6 en le configurant avec un *Prescaler* = 8000 -1 et un *Counter Period (Autoreload Register – 16 bits)* = 65535-1

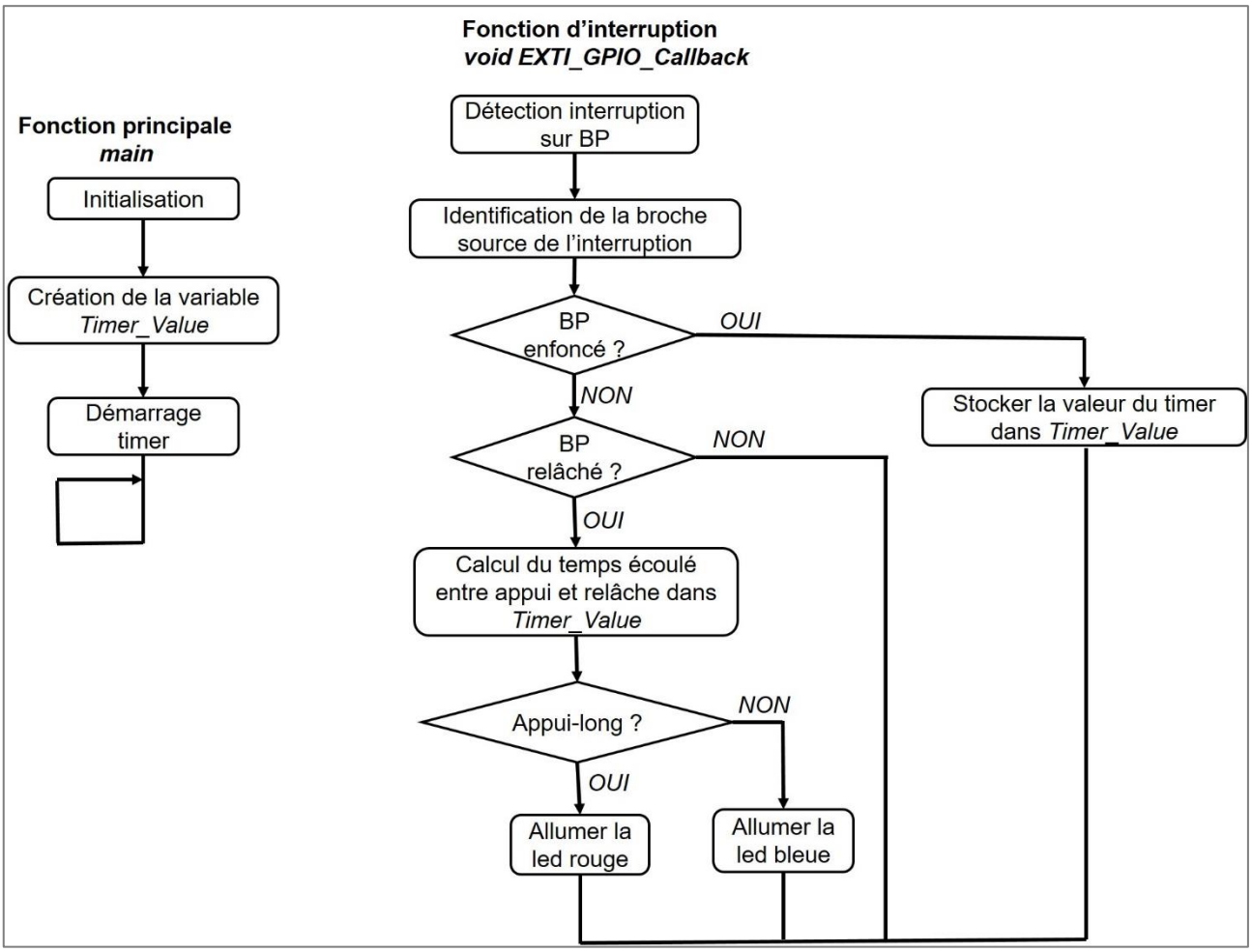


- Sur l'interface CubeMX dans l'onglet « Clock Configuration », vérifier que l'horloge du timer appelée « *APB1 Timer Clock* »est bien à une valeur de 8 MHz



- Générer le code d'initialisation

L'algorigramme ci-dessous présenta le programme à créer pour mettre en œuvre la détection d'un appui-court ou long et d'allumer la led en conséquence.



La programmation consiste à entrer dans la fonction d'interruption uniquement lorsque l'utilisateur appui ou relâche le bouton-poussoir. Cette fonction d'interruption sera à programmer. Elle doit **impérativement** s'appeler : *HAL_GPIO_EXTI_Callback*

La fonction principale *main* ne fait quasiment rien. Tout se passe dans cette fonction d'interruption.

La fonction *main* crée la variable *Timer_Value* et démarre le timer.

La fonction *HAL_GPIO_EXTI_Callback*

- *détecte l'appui et la relâche du bouton-poussoir*
- *calcule l'intervalle de temps (à l'aide du timer) entre l'appui et la relâche du bouton-poussoir*
- *allume la led RG en fonction de si l'appui est court ou long*

Voici les lignes de code associées

- Programmation de la fonction *main*

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */
  uint16_t Timer_Value = 0;  // Variable pour stocker la durée de l'appui
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  HAL_TIM_Base_Start(&htim6);      // Démarrage du timer
  while (1)
  {
    /* USER CODE END WHILE */

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

- Programmation de la fonction *EXTI_GPIO_Callback entre les basiles /*USER CODE BEGIN 4 */* et */* USER CODE END 4 */ vers la ligne 240.*

```
/* USER CODE BEGIN 4 */
////////////////////////////////////////////////////////////////////////////
////////// Programme d'interruption executée lors de l'appuie sur le BP ////////
////////////////////////////////////////////////////////////////////////////
void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)
{
	if (GPIO_Pin == BP_Pin)   // Si interruption vient du BP
	{
		if(HAL_GPIO_ReadPin(BP_GPIO_Port, BP_Pin)==1)// Si appuie sur BP OK
		{
			__HAL_TIM_SetCounter(&htim6, 0);// initialisation du compteur
			Timer_Value = 0;
		}
		else if (HAL_GPIO_ReadPin(BP_GPIO_Port, BP_Pin)==0) // Si relache du BP OK
		{
			Timer_Value = __HAL_TIM_GetCounter(&htim6);// détermination de la durée
de l'appui stockée dans Timer_Value
			if (Timer_Value > 1000)   // Si appui long
			{
				HAL_GPIO_WritePin(GPIOB,  Red_Pin, GPIO_PIN_RESET); // Led Red
OFF
				HAL_GPIO_WritePin(Blue_GPIO_Port,  Blue_Pin, GPIO_PIN_SET);//
Led Blue ON

			}
			else if (Timer_Value < 1000 && Timer_Value > 0)
			{
				HAL_GPIO_WritePin(GPIOB,  Red_Pin, GPIO_PIN_SET);// Led Red ON
				HAL_GPIO_WritePin(Blue_GPIO_Port,  Blue_Pin, GPIO_PIN_RESET) ;//
Led Blue OFF
			}
			else {
				HAL_GPIO_WritePin(GPIOB,  Red_Pin, GPIO_PIN_RESET); // Led Red
OFF
				HAL_GPIO_WritePin(Blue_GPIO_Port,  Blue_Pin, GPIO_PIN_RESET) ;//
Led Blue OFF
			}

		}
		else {

		}
	}
}
/* USER CODE END 4 */
```

La fonction *EXTI_GPIO_Callback* est à coder entièrement entre les balises */*USER CODE BEGIN 4 */* et */* USER CODE END 4 */*

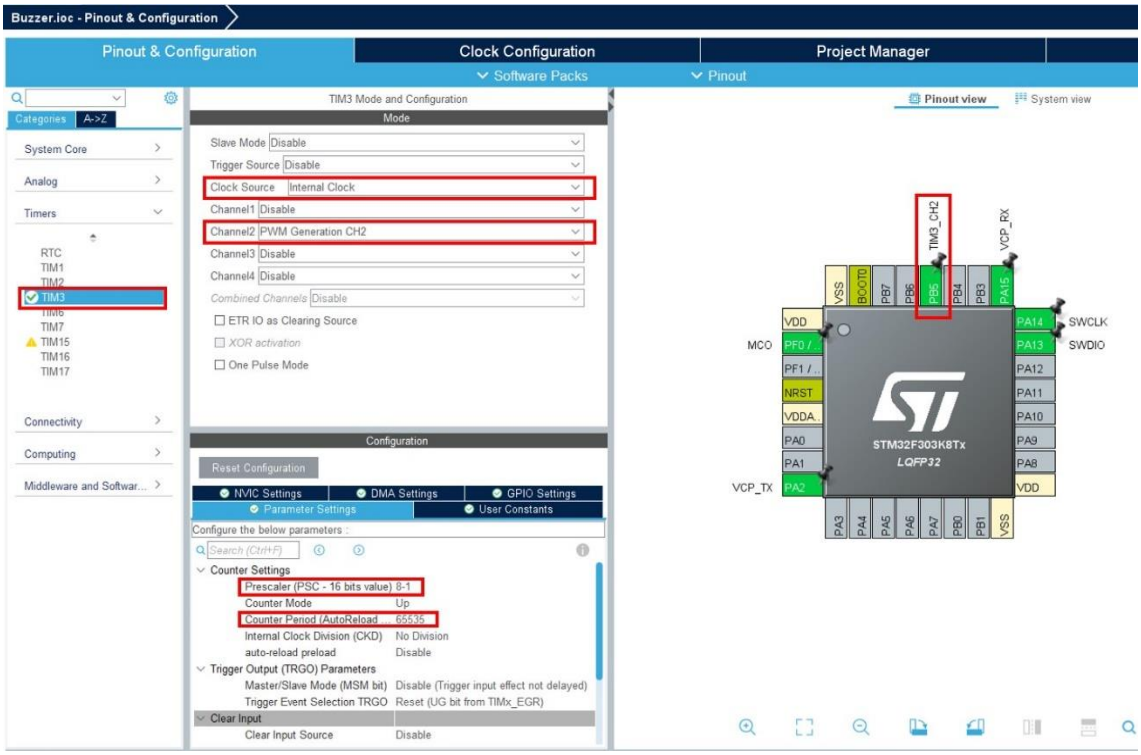
Pour plus d'aide, vous pouvez consulter la vidéo suivante : https://www.youtube.com/watch?v=VfbW6nfG4kw&ab_channel=Digi-Key

3. Commande du buzzer 245-6528

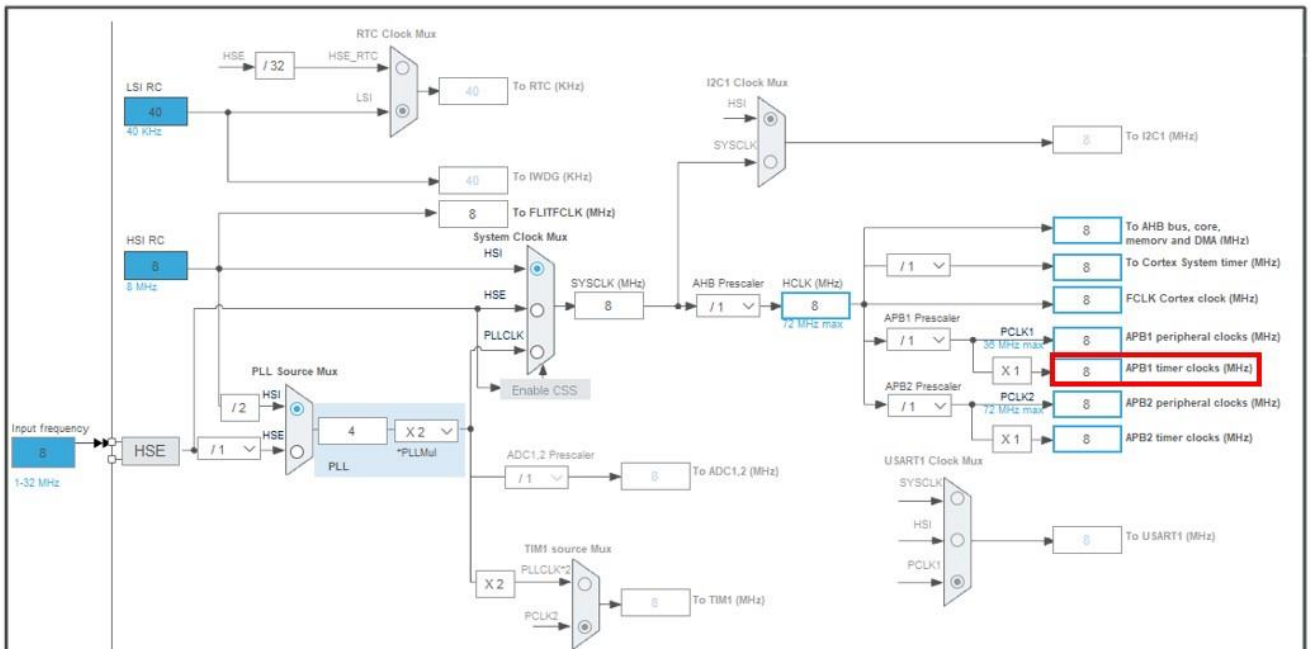
L'objectif de cette partie est de présenter la méthodologie à suivre pour commander un buzzer 245-6528 afin de générer un son audio.

Cet exemple est présenté pour une carte de développement NUCLEO-F303K8, mais la méthodologie est la même pour un autre type de microcontrôleur de la famille STM32.
Dans cet exemple, l'objectif est de jouer une gamme de musique par le buzzer 245-6528.
Pour faire cela, il sera nécessaire d'utiliser un *timer* pour générer un signal PWM.

- Créer un nouveau projet avec la NUCLEO-F303K8
- Sur l'interface CubeMX, initialiser la broche du microcontrôleur PB5 en TIM3_CH2
- Sur l'interface CubeMX, configurer le Timer 3 avec les paramètres suivants :
  - Clock Source = Internal Clock
  - Channel 2 : PWM Generation CH2
  - Prescaler = 8-1
  - Counter Period = 65535



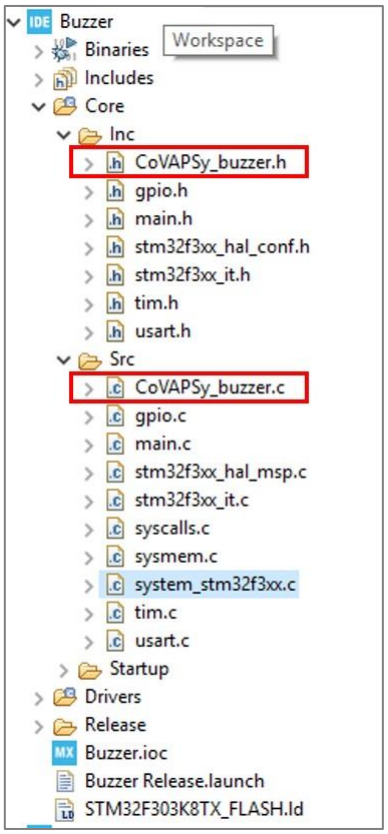
- Sur l'interface CubeMX dans l'onglet « Clock Configuration », vérifier que l'horloge du timer appelée « *APB1 Timer Clock* » est bien à une valeur de 8 MHz



- Générer le code d'initialisation
- Câbler le buzzer avec la NUCLEo-F303K8 en utilisant les bonnes broches.

Pour commander le buzzer, il faut installer et utiliser la bibliothèque CoVAPsy_Buzzer. Cette bibliothèque est disponible sur le réseau pédagogique.

- Copier le fichier « CoVAPSy_buzzer.h » dans le dossier Core=>Inc et le fichier « CoVAPSy_buzzer.c » dans le dossier « Core => Src »



- Ajouter la bibliothèque dans le code principal (fichier main.c) en ajoutant la ligne de code #include "CovaPSY_buzzer.h" au début du programme

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "CoVAPSy_buzzer.h"
/* USER CODE END Includes */
```

Voici les lignes de code à ajouter dans la fonction *main* pour faire jouer une gamme complète à votre buzzer puis jouer un bip grave, puis un bip grave puis un bip aigu et enfin un bip aigu.

- Dans la fonction *main*, ajouter les lignes de code suivantes :

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    buzzer_start();         // Joue une gamme de note
    buzzer-gamme();
    HAL_Delay(1000);

    buzzer_start();         // Bip grave
    buzzer_start_frequency_Hz(NOTE_DO3);
    HAL_Delay(100);
    buzzer_stop();
    HAL_Delay(100);

    buzzer_start();         // Bip grave
    buzzer_start_frequency_Hz(NOTE_DO3);
    HAL_Delay(100);
    buzzer_stop();
    HAL_Delay(100);

    buzzer_start();         // Bip aigue
    buzzer_start_frequency_Hz(NOTE_DO4);
    HAL_Delay(100);
    buzzer_stop();
    HAL_Delay(100);

    buzzer_start();         // Bip aigue
    buzzer_start_frequency_Hz(NOTE_DO4);
    HAL_Delay(100);
    buzzer_stop();
    HAL_Delay(100);
  /* USER CODE END WHILE */

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

- Compiler le programme avec le bouton « Build »
- Lancer l'exécution du programme avec le bouton « run ».

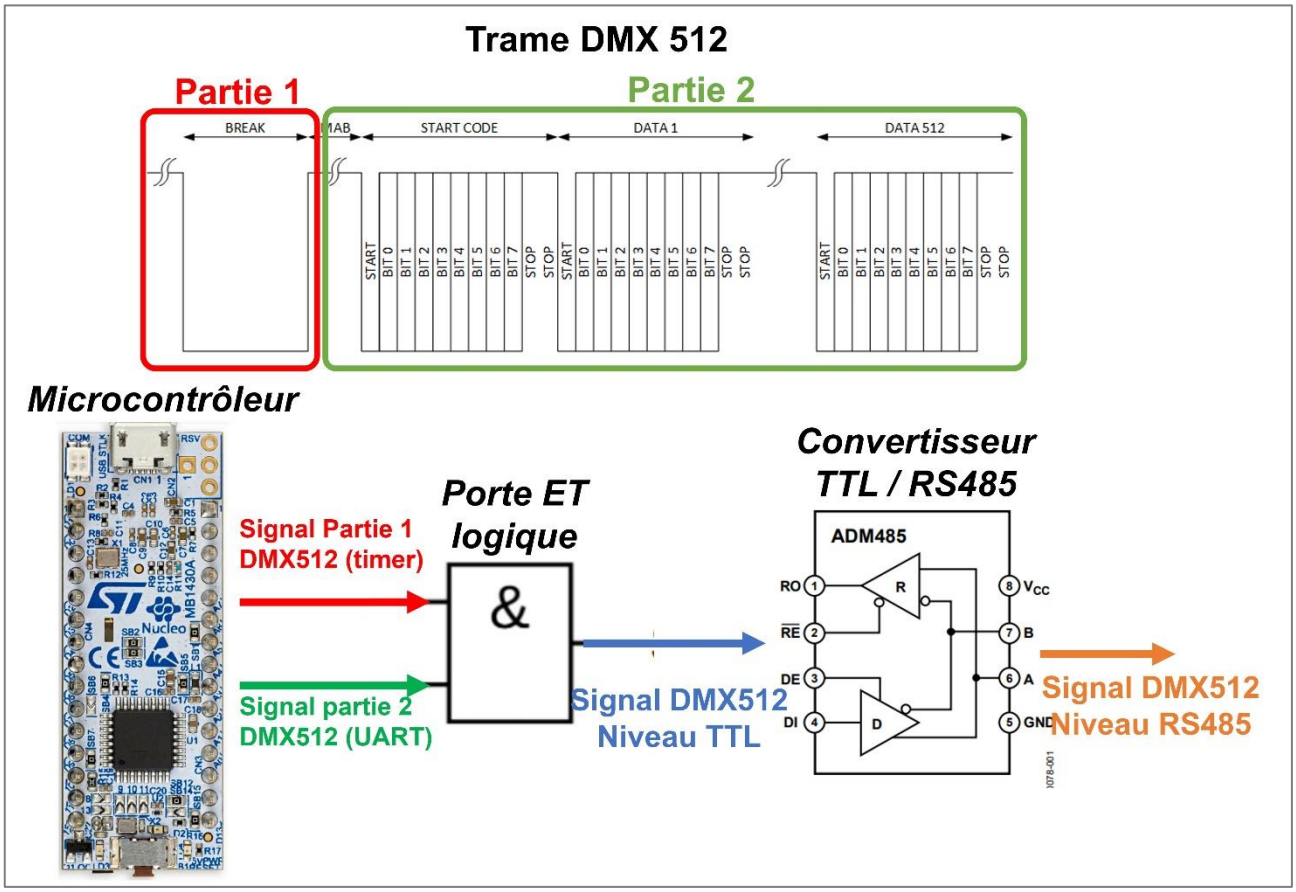
Une jolie mélodie doit être jouée.

IX. PROTOCOLE DMX 512

L'objectif de cette partie est de présenter une bibliothèque en langage C mise-à-votre disposition pour transmettre des trames compatibles avec le protocole DMX 512.

1. Rappel sur la trame DMX 512

La trame DMX 512 est structurée selon le schéma ci-dessous.
Le débit binaire est de 250 kbit/s.
Le trame DMX permet de commander jusqu'à 512 canaux DMX.

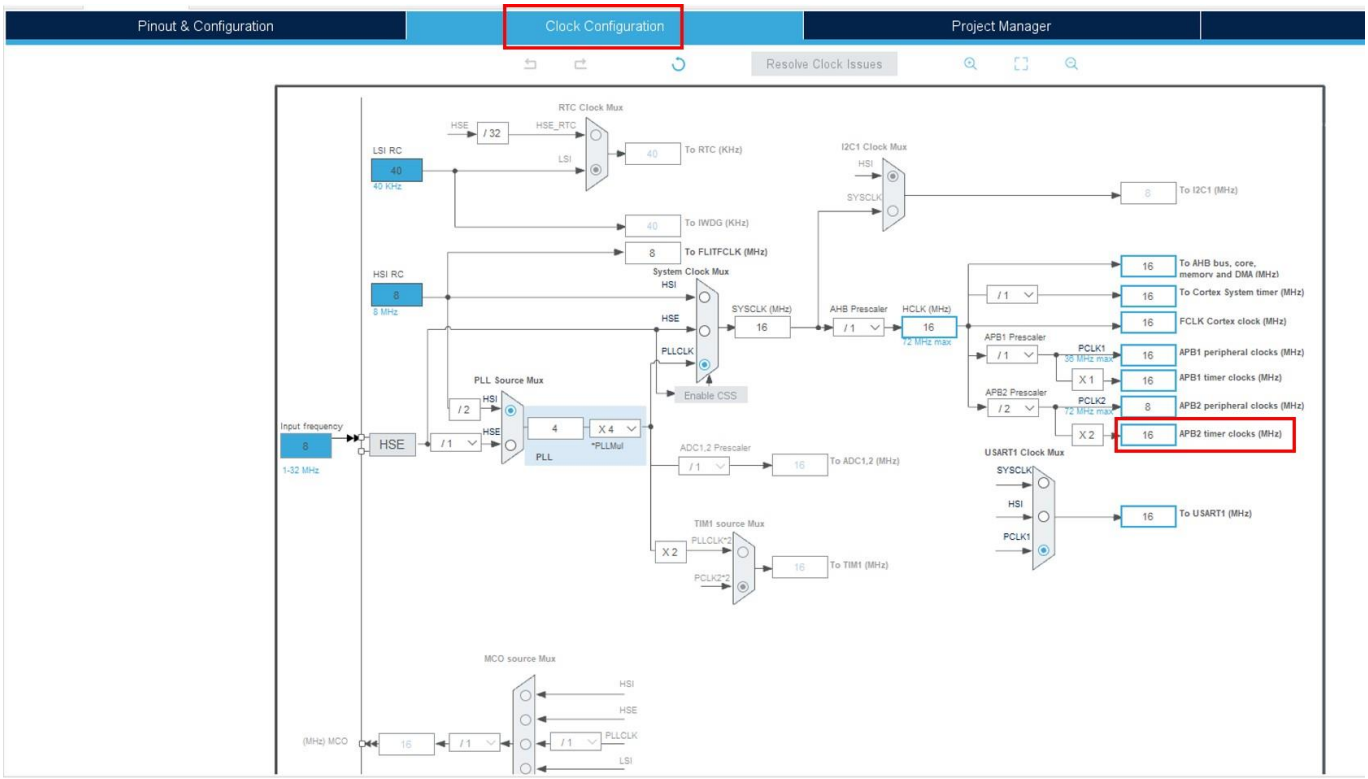


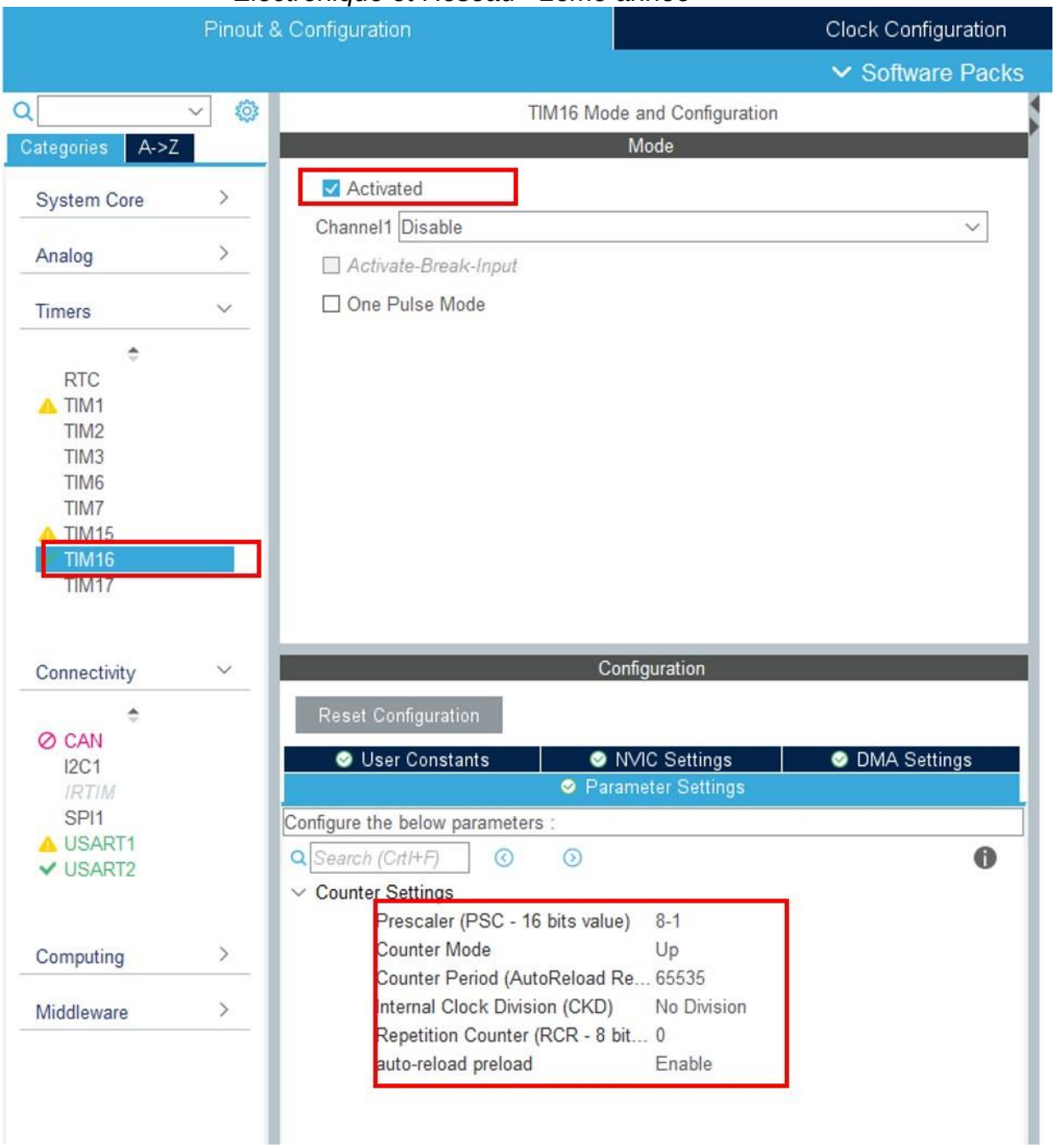
La première partie de la trame DMX512 sera générée par une broche GPIO Output à l'aide d'un compteur (timer).
La deuxième partie de la trame DMX512 sera générée par une liaison UART.
Un ET logique sera effectué entre les 2 signaux afin de générer la trame DMX512 au niveau TTL.

2. Installation de la bibliothèque DMX 512

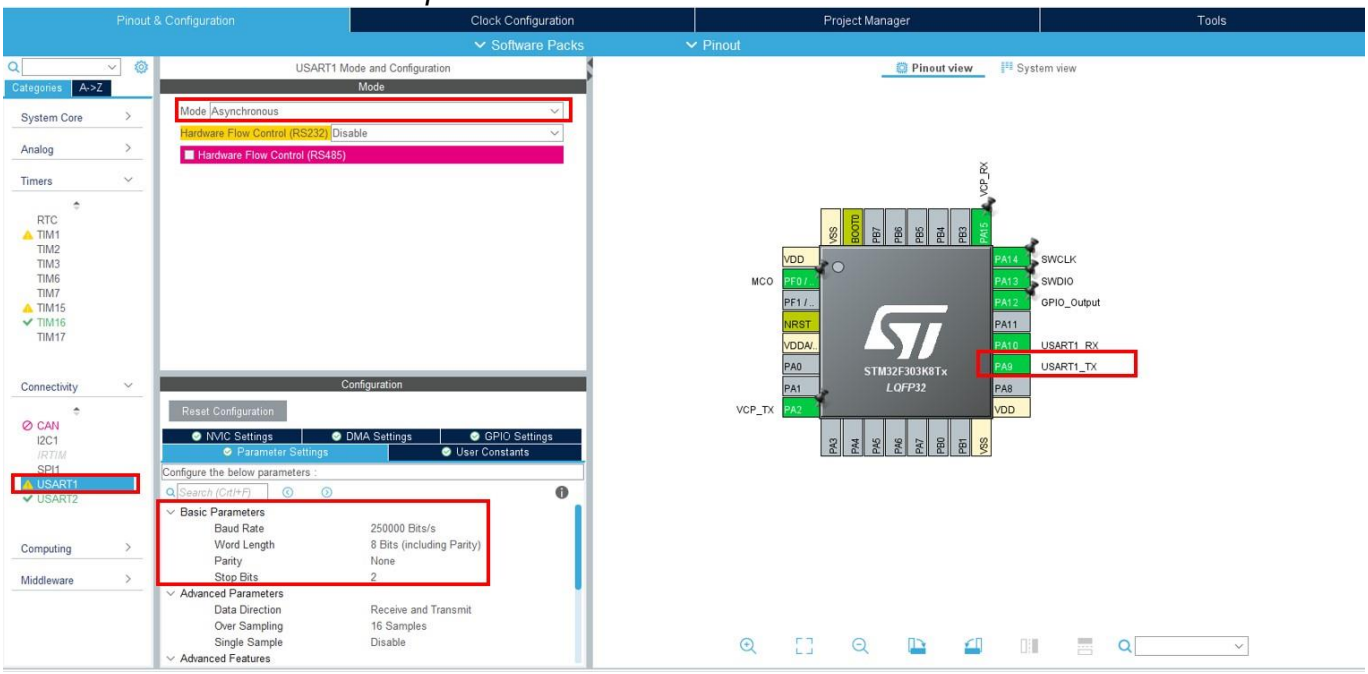
Pour pouvoir utiliser cette bibliothèque DMX 512, la procédure à suivre est la suivante :

- Sous le logiciel STM32CubeMX, création d'une sortie numérique (GPIO) qui servira pour transmettre la première partie de la trame DMX 512 au niveau TTL
- Sous le logiciel STM32CubeMX, création d'un timer TIM16 avec une base de temps de 500ns avec une horloge de 16 MHz

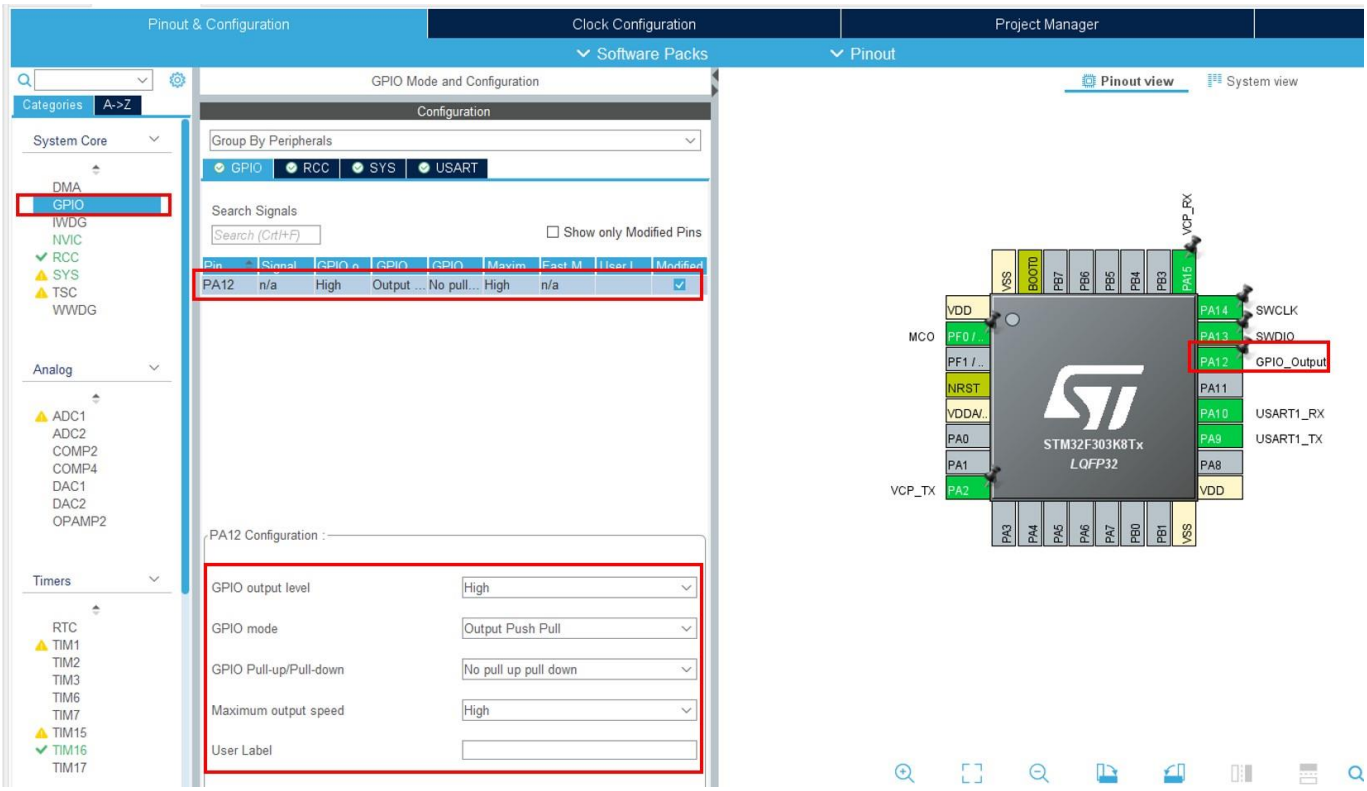




- Sous le logiciel STM32CubeMX, création d'une liaison UART configurée comme ci-dessous

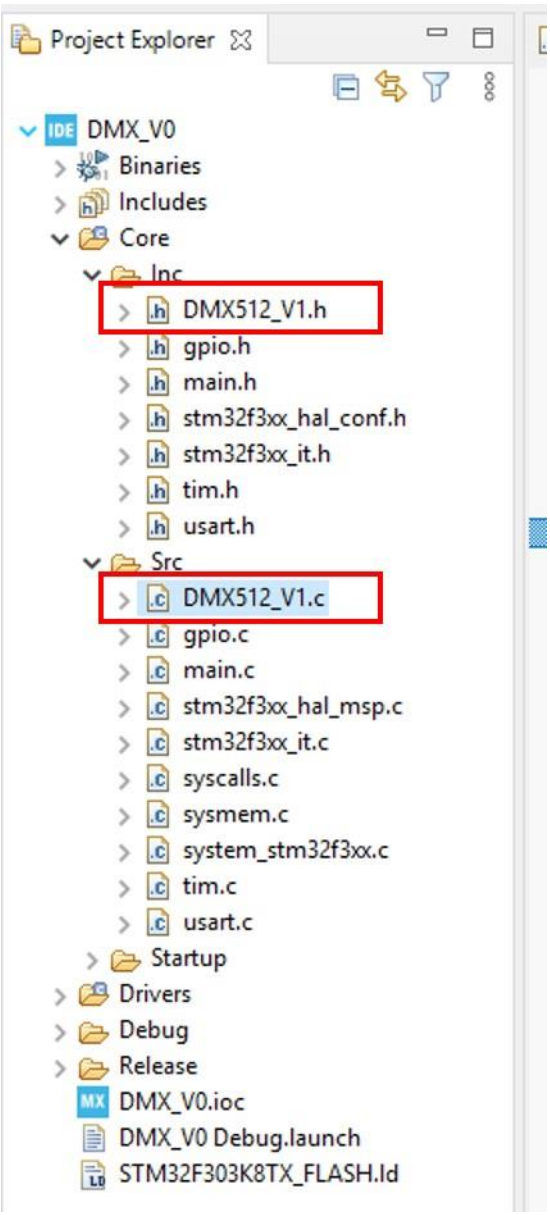


- Sous le logiciel STM32CubeMX, création d'une broche GPIO Output configurée comme ci-dessous



3. Programmation en C pour mettre en œuvre la bibliothèque DMX512

- Ajout de la bibliothèque DMX 512
  - Copier le fichier DMX512_V1.h dans le dossier Core/Inc de votre projet STM32CubeIDE
  - Copier fichier DMX512_V1.c dans le dossier Core/Src de votre projet STM32CubeIDE



- Modifier l'en-tête du fichier *main.c* afin d'ajouter la bibliothèque DMX 512 :

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "DMX512_V1.h"
/* USER CODE END Includes */
```

- Modifier la fonction principale *main* afin d'utiliser la bibliothèque DMX512. L'exemple ci-dessous met le canal DMX 512 n° 1 à une valeur de 255.

```
/* USER CODE BEGIN 2*/
uint8_t Data_TX[4]={0,255,0,0} ; // Déclaration et Initialisation de la donnée Data_TX

HAL_TIM_Base_Start(&htim16);    // Démarrage du timer 16
/* USER CODE END 2*/


/* USER CODE BEGIN WHILE-*/
While (1)
{
     HAL_GPIO_WritePin(GPIOA, GPIO_Pin_12, GPIO_PIN_RESET) ;   // Broche PA12 à l'état bas
     delay_us(100) ;   // Attente 100us
     HAL_GPIO_WritePin(GPIOA, GPIO_Pin_12, GPIO_PIN_SET) ;   // Broche PA12 à l'état haut
     delay_us(1) ;   // Attente 1us
     HAL_UART_Transmit(&huart1, Data_Tx, sizeof(Data_Tx), 100); // Transmission en UART de
Data_Tx
    HAL_Delay(200) ;   // Attente de 200ms avant d'émettre une nouvelle trame DMX512

/* USER CODE END WHILE-*/

}
```

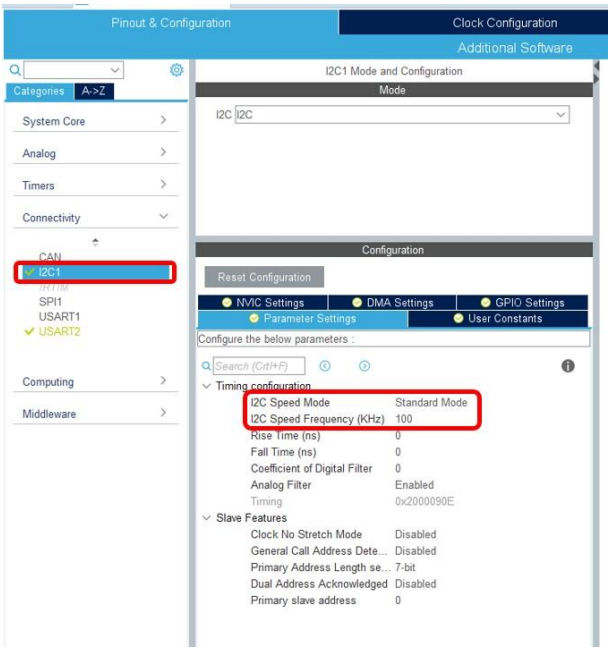
X. CAPTEUR DE TEMPÉRATURE MCP9808

L'objectif de cette partie est de présenter l'utilisation d'une bibliothèque STM32 pour le capteur de température MICROCHIP – MCP9808 ou l'outil de développement associé.

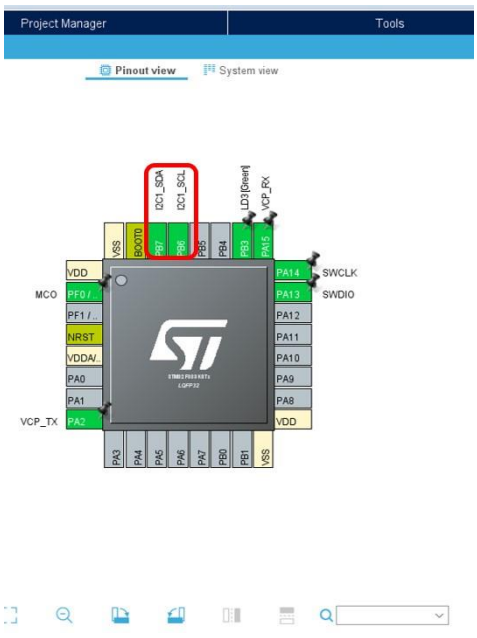
1. Installation de la bibliothèque *CapteurTemperature_MCP9808*

Pour pouvoir utiliser cette bibliothèque *CapteurTemperature_MCP9808*, la procédure à suivre est la suivante :

- Sous le logiciel STM32CubeMX, création d'une liaison I2C en mode standard à une fréquence de 100kHz



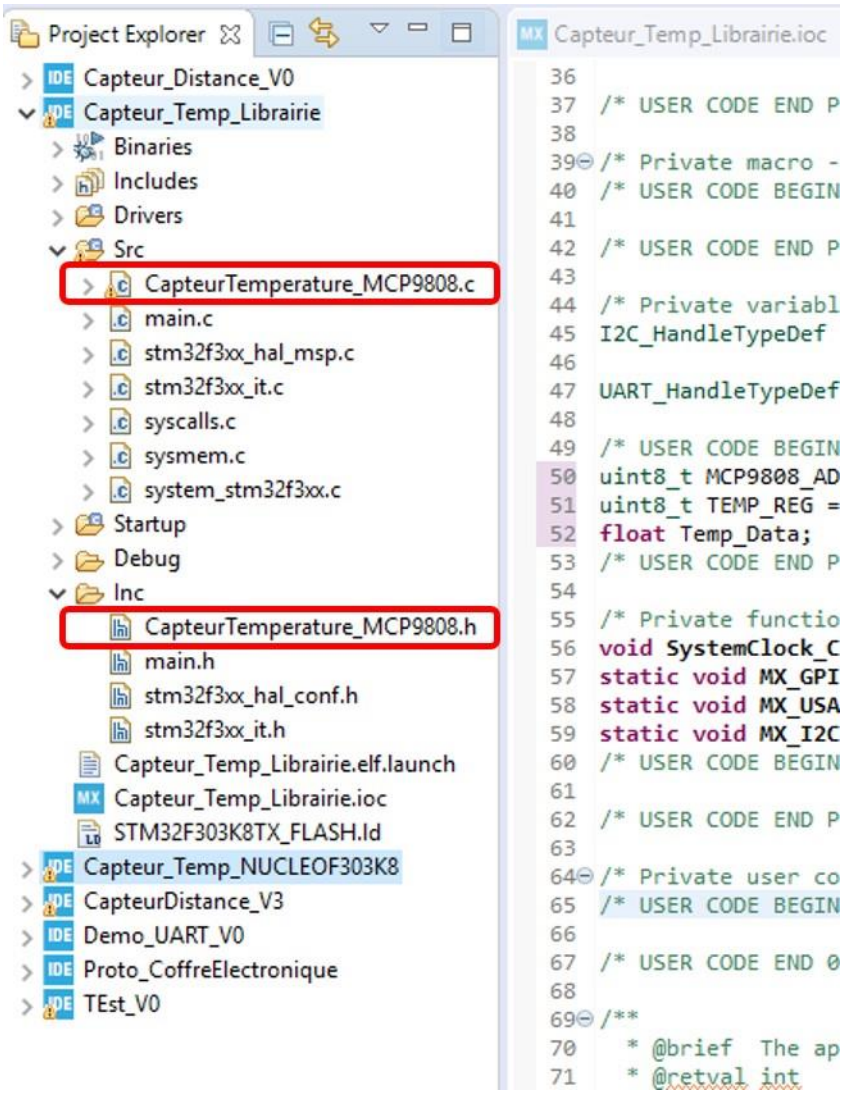
- Identifier les broches utilisées pour les signaux I2C_SDA (data) et I2C_SCL (Clock)



- Sauvegarder le projet afin de lancer la génération du code d'initialisation s'effectue.

- Câbler correctement le capteur de température MCP9808 au microcontrôleur en fonction des broches identifiées et de la documentation technique du capteur de température MCP9808.

- Dans le fichier répertoire de votre projet STM32CudeIDE, ajouter la libraire MCP9808.
  - Copier le fichier *CapteurTemperature_MCP9808.h* dans le dossier Inc de votre projet STM32CubeIDE
  - Copier le fichier *CapteurTemperature_MCP9808.c* dans le dossier Src de votre projet STM32CubeIDE



2. Programme à faire pour utiliser la bibilothèque *CapteurTemperature_MCP9808*

- Modifier le fichier *main.c* afin d'inclure la bibliothèque *CapteurTemperature_MCP9808* dans le programme principal :

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
 #include "i2c.h"
 #include "usart.h"
 #include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "CapteurTemperature_MCP9808.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
uint8_t MCP9808_ADDR = XXX;         //I2C Address of the MCP9808 Temperature Sensor
uint8_t TEMP_REG = XXX;             //I2C Register associated to the Temperature Data
float Temp_Data;                    //Variable to store the Temperature Data
```

Attention : A vous d'identifier les valeurs des variables MCP9808_ADDR et TEMP_REG. Dans l'exemple ci-dessus, il faut donc modifier les « XXX » par des valeurs issues de la datasheet du composant MCP9808.

- Ajouter les lignes de code suivantes dans le fichier *main.c* dans la boucle while :

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        Temp_Data = Get_Temp(MCP9808_ADDR, TEMP_REG);           // Get the Temperature
from the MCP9808 Sensor
    /* USER CODE END WHILE */

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

Et voilà, le programme utilisant la bibliothèque *CapteurTemperature*_MCP9808 est opérationnel !

La valeur de la température mesurée par le capteur de température MCP9808 est stockée dans la variable *Temp_Data.*

- Compiler le programme en mode débug.
- Visualiser la variable *Temp_Data.*

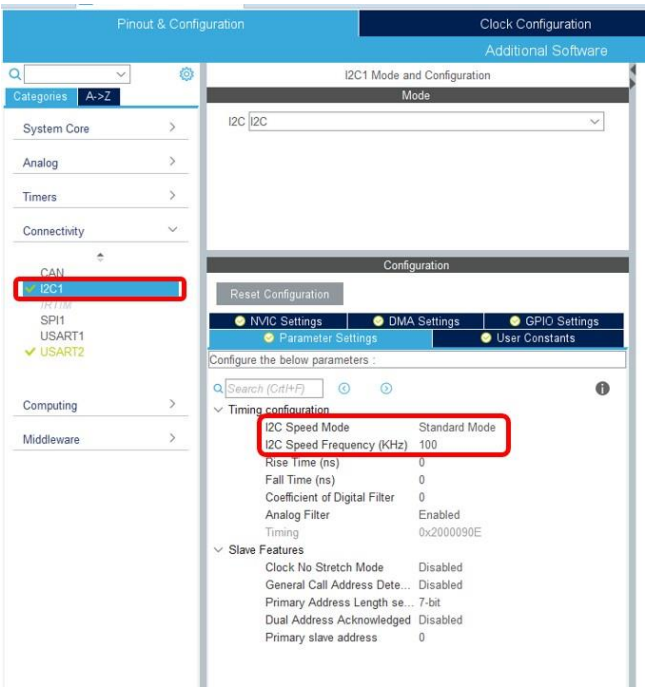
XI. CAPTEUR DE DISTANCE VCNL3030X01-GS08

L'objectif de cette partie est de présenter le traitement de donnée spécifique pour le capteur de distance VISHAY– VCNL3030X01-GS08 ou la maquette pédagogique associé.

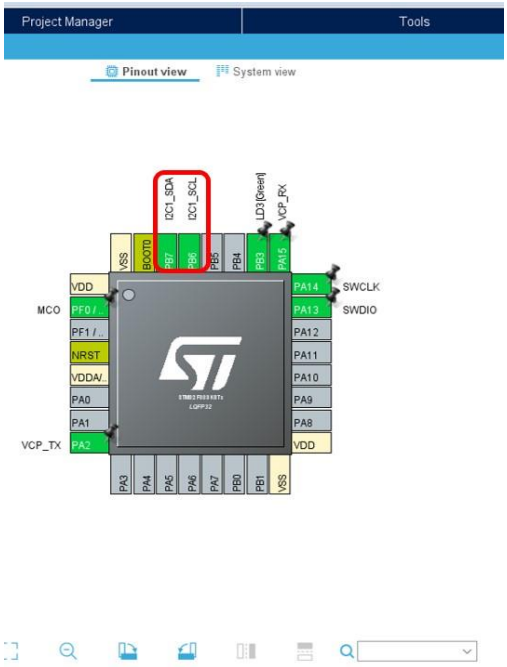
1. Installation de la bibliothèque *CapteurDistance_VCNL3030X01*

Pour pouvoir utiliser cette bibliothèque *CapteurDistance_VCNL3030X01*, la procédure à suivre est la suivante :

- Sous le logiciel STM32CubeMX, création d'une liaison I2C en mode standard à une fréquence de 100kHz



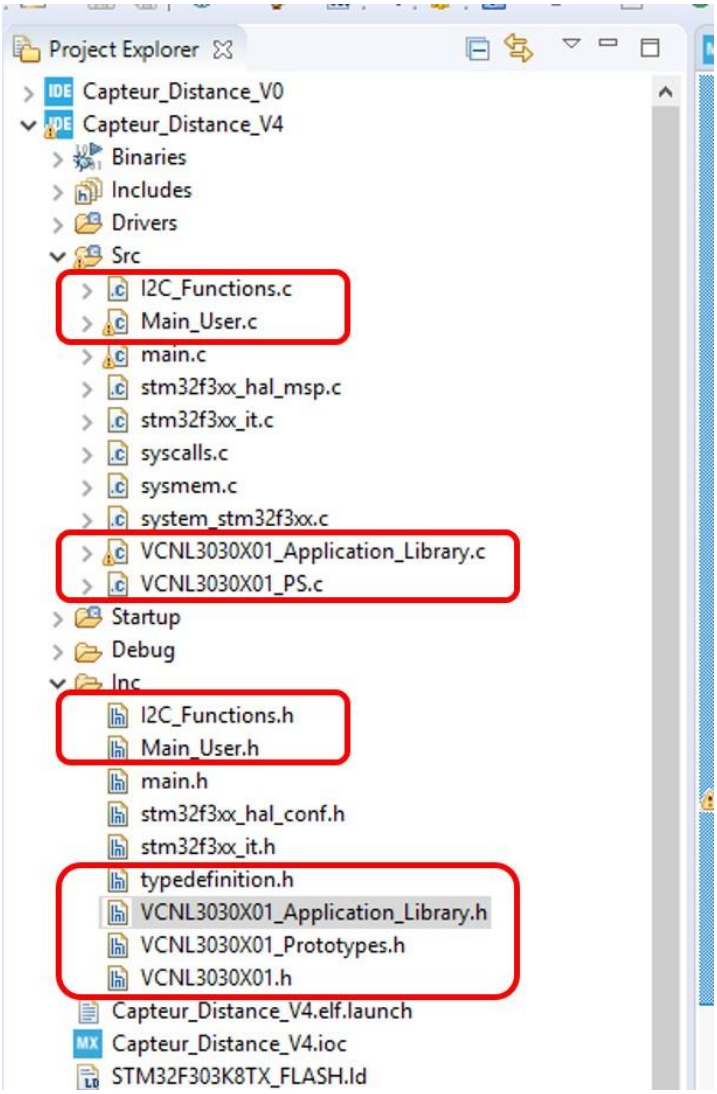
- Identifier les broches utilisées pour les signaux I2C_SDA (data) et I2C_SCL (Clock)



- Sauvegarder le projet afin de lancer la génération du code d'initialisation s'effectue.

- Câbler correctement le capteur de distance VCNL3030X01 au microcontrôleur en fonction des broches identifiées et de la documentation technique du capteur de distance VCNL3030X01.

- Dans le fichier répertoire de votre projet STM32CudeIDE, ajouter la libraire *CapteurDistance_VCNL3030X01*.
  - Copier les fichiers suivants dans le dossier Inc de votre projet STM32CubeIDE :
    - *I2C_Functions.h*
    - *Main_User.h*
    - *typedefinition.h*
    - *VCNL3030X_Application_Library.h*
    - *VCNL3030X01_Prototypes.h*
    - *VCNL3030X01.h*
  - Copier les fichiers suivants dans le dossier Src de votre projet STM32CubeIDE :
    - *I2C_Functions.c*
    - *Main_User.c*
    - *VCNL3030X_Application_Library.c*
    - *VCNL3030X01_PS.c*



1. Programme pour utiliser la bibliothèque *CapteurDistance_VCNL3030X01*.

Le fichier mère de la bibliothèque *CapteurDistance_VCNL3030X01* est le fichier *Main_User.h*.

- Ajouter la bibliothèque *CapteurDistance_VCNL3030X01* (*Main_User.h*.) dans le programme principal *main.c*:

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "i2c.h"
#include "usart.h"
#include "gpio.h"


/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "Main_User.h"            //Ajout de la bibliothèque CapteurDistance_VCNL3030X01

/* USER CODE END Includes */
```

- Déclarer la variable *Value* qui stockera la donnée de distance du capteur dans le programme principal *main.c*:

```
/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
uint16_t Value;     //Déclaration de la variable pour stocker la distance mesurée par le capteur
/* USER CODE END PV */
```

- Ajouter les lignes de code suivantes dans le fichier *main.c* afin d'initialiser le capteur de distance VCNL3030X puis de récupérer la distance mesurée :

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
INIT_VCNL3030X01();     // Initialisation du capteur de distance VCNL3030X01
while (1)
{
Value = Get_Data();// Récupérer la mesure de la distance par le capteur VCNL3030X01
   HAL_Delay(100);
   /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
 }
```

Et voilà, le programme utilisant la bibliothèque *CapteurDistance _VCNL3030X01* est opérationnel !

La valeur de la température mesurée par le capteur de distance VCNL3030X1 est stockée dans la variable *Value.*

- Compiler le programme en mode débug.
- Visualiser la variable *Value*

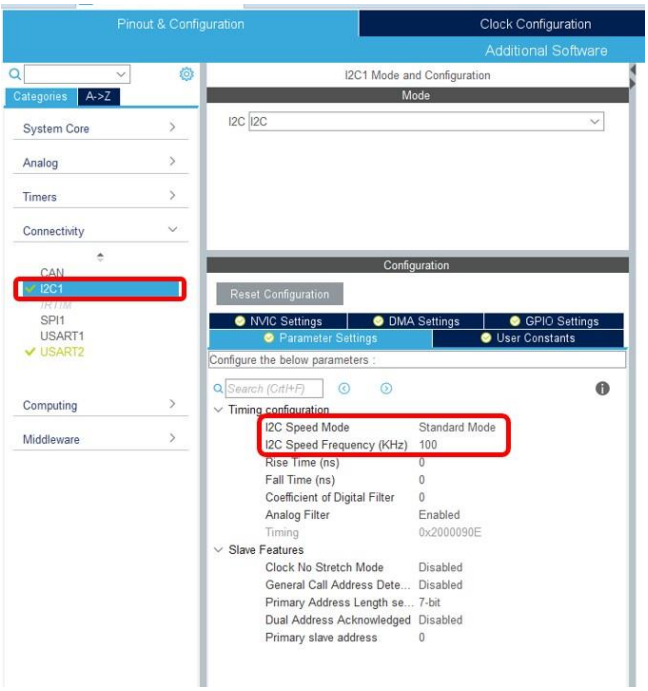
XII. CATPEUR DE DISTANCE SRF10

L'objectif de cette partie est de présenter le traitement de donnée spécifique pour le capteur de distance SRF10

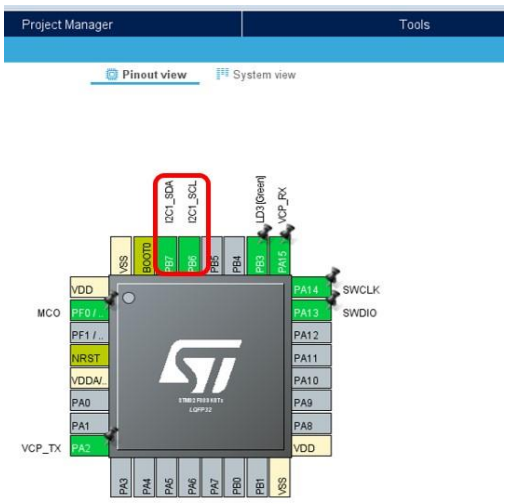
1. Installation de la bibliothèque *Telemetre_SRF10*

Pour pouvoir utiliser cette bibliothèque *Telemetre_SRF10*la procédure à suivre est la suivante :

- Sous le logiciel STM32CubeMX, création d'une liaison I2C en mode standard à une fréquence de 100kHz



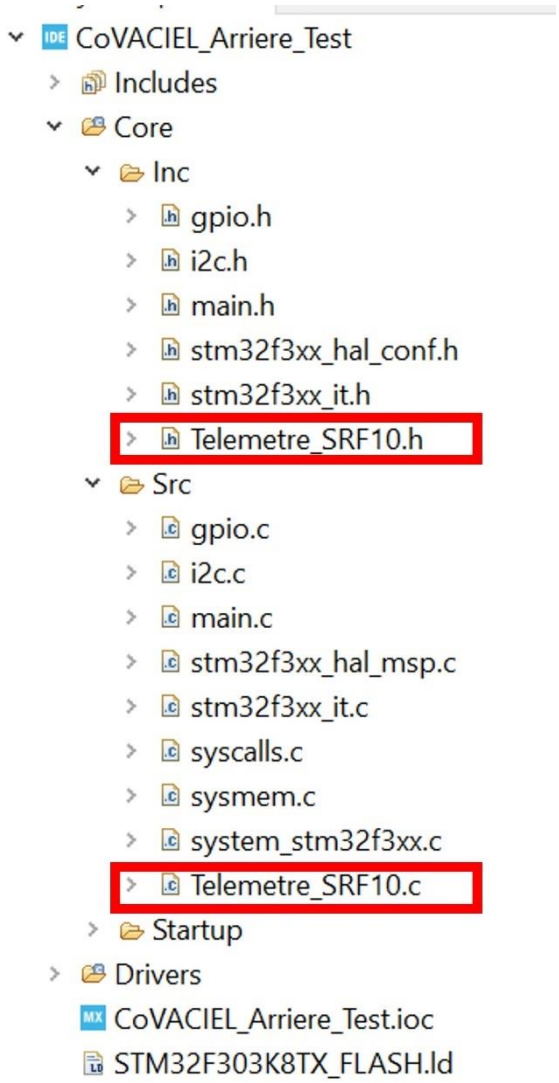
- Identifier les broches utilisées pour les signaux I2C_SDA (data) et I2C_SCL (Clock)



- Sauvegarder le projet afin de lancer la génération du code d'initialisation s'effectue.

- Câbler correctement le capteur de distance SRF10 au microcontrôleur en fonction des broches identifiées et de la documentation technique du capteur de distance SRF10.

- Dans le fichier répertoire de votre projet STM32CudeIDE, ajouter la bibliothèque *Telemetre_SRF10*.
  - Copier les fichiers suivants dans le dossier Inc de votre projet STM32CubeIDE :
    - *Telemetre_SRF10.h*
  - Copier les fichiers suivants dans le dossier Src de votre projet STM32CubeIDE :
    - *Telemetre_SRF10.c*



2. Programmation pour utiliser la bibliothèque *Telemetre_SRF10*,

- Ajouter la bibliothèque *Telemetre_SRF10* en modifiant le fichier main.c dans le programme principal sous le commentaire `/* USER CODE BEGIN Includes */`.

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "Telemetre_SRF10.h"
/* USER CODE END Includes */
```

- Déclarer une variable de type entier 16 bits afin de stocker la donnée mesurée par le capteur de distance SRF10. Déclarer cette variable dans le fichier main.c sous le commentaire `/* USER CODE BEGIN PV */`.

```
/* USER CODE BEGIN PV */
int16_t Distance_Telemetre;                    //Distance mesurée par le capteur SRF10
/* USER CODE END PV */
```

- Ajouter la ligne de code suivantes dans le fichier *main.c* afin de récupérer la distance mesurée par le SRF10 :

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    Distance_Telemetre = Mesure_Telemetre();// Récupérer la mesure de la distance par le capteur SRF10
     HAL_Delay(100);
     /* USER CODE END WHILE */
      /* USER CODE BEGIN 3 */
   }
```

Et voilà, le programme utilisant la bibliothèque *Telemetre_SRF10* est opérationnel !

La valeur de la distance mesurée par le capteur de distance SRF10 est stockée dans la variable *Distance_Telemetre.*

- Compiler le programme en mode débug.
- Visualiser la variable *Distance_Telemetre*

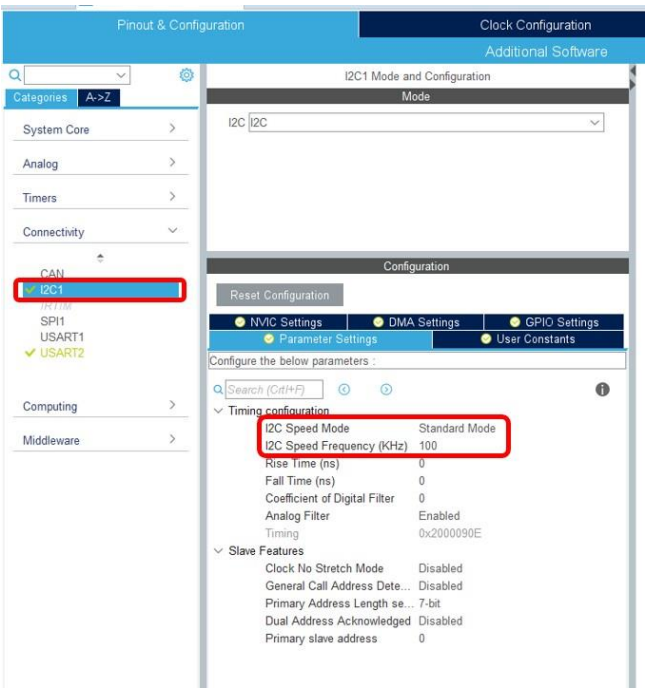
XIII. AFFICHEUR OLED

L'objectif de cette partie est de présenter la programmation pour commander l'afficheur OLED.

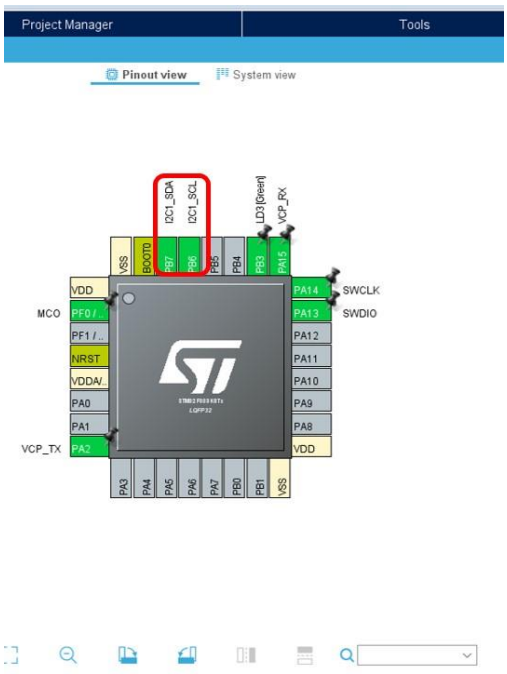
1. Installation de la bibliothèque *OLED*

Pour pouvoir utiliser cette bibliothèque *OLED*, la procédure à suivre est la suivante :

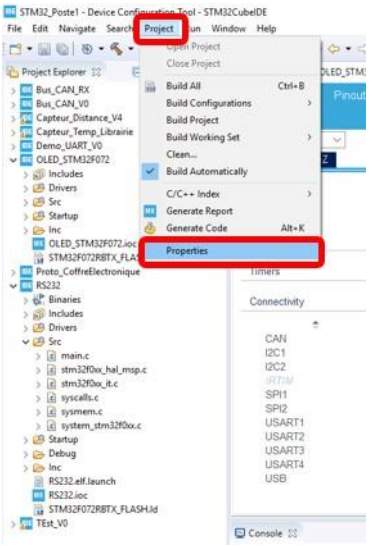
- Sous le logiciel STM32CubeMX, création d'une liaison I2C en mode standard à une fréquence de 400kHz



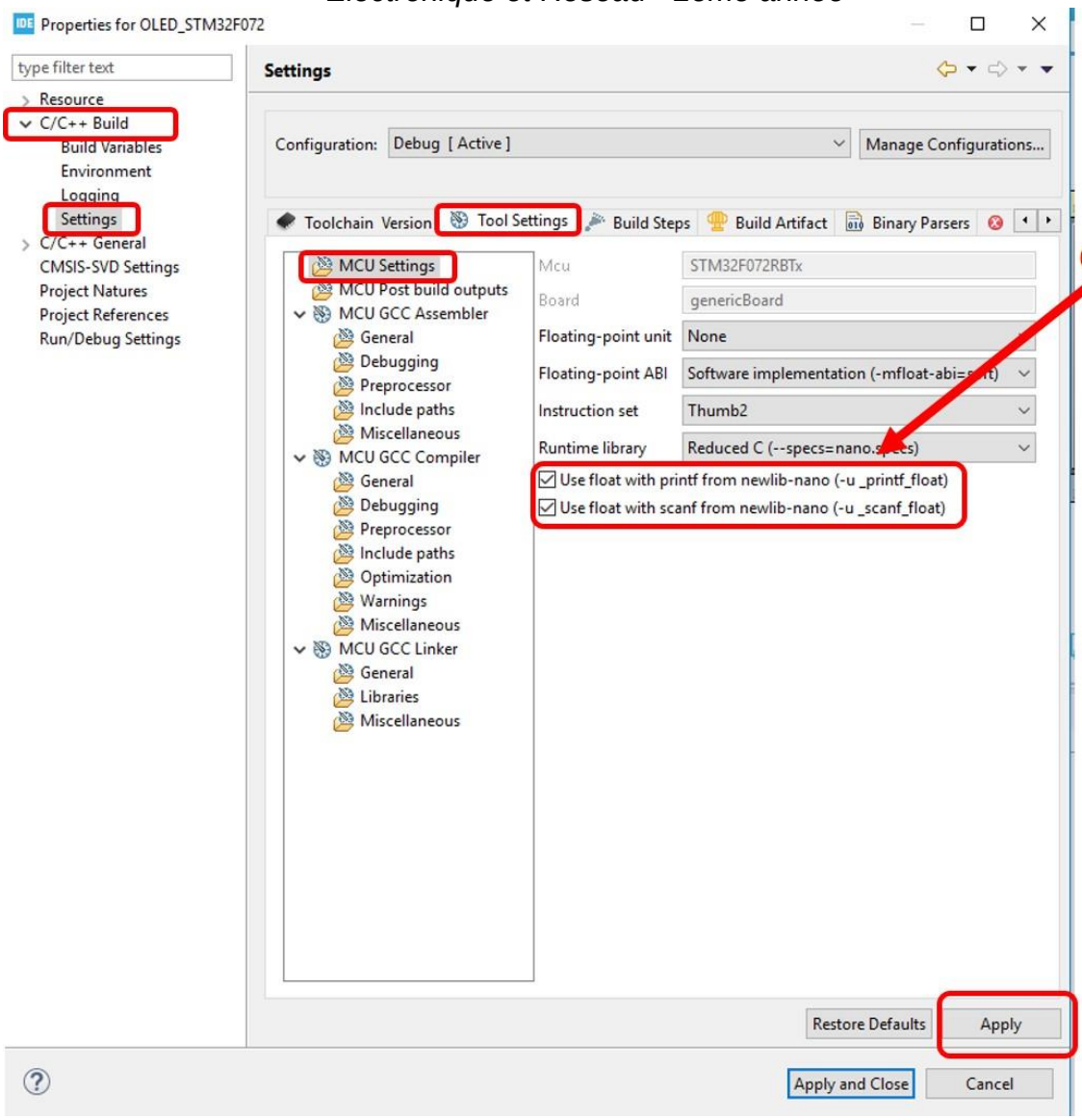
- Identifier les broches utilisées pour les signaux I2C_SDA (data) et I2C_SCL (Clock)



- Modifier les propriétés de votre projet pour prendre en compte les caractères ASCII
  - Cliquer sur Projet => Properties



  - En allante dans « C/C++ Build => Settings => Tool Settings=> MCU Settings », cocher les 2 cases suivantes :

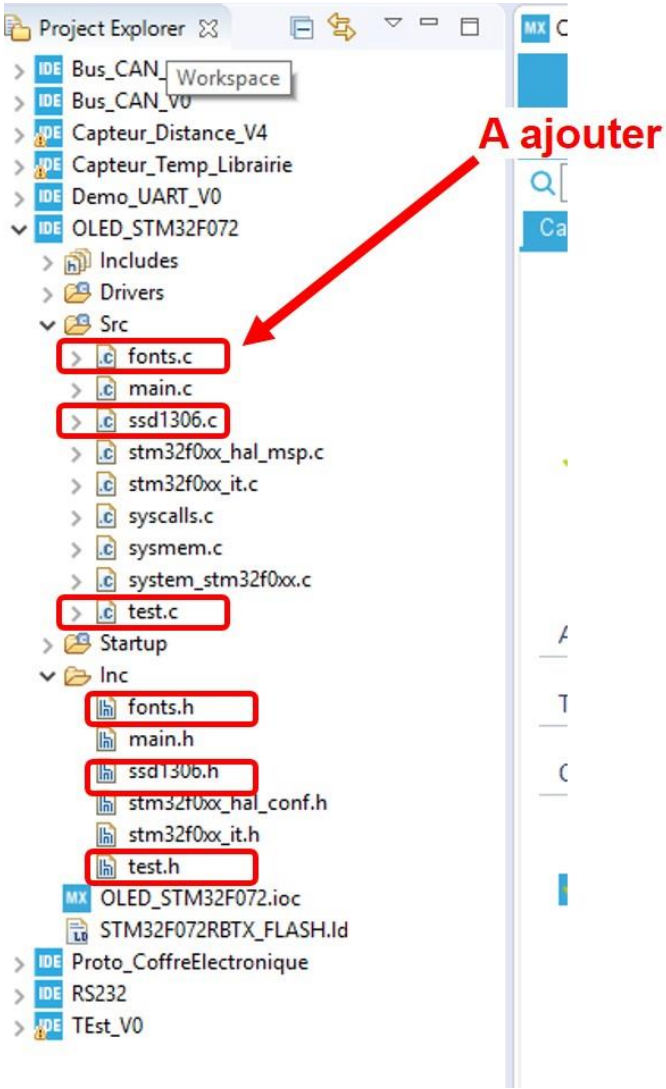


- Sauvegarder le projet afin de lancer la génération du code d'initialisation s'effectue.

- Câbler correctement l'afficheur OLED au microcontrôleur en fonction des broches identifiées.

Attention: Le module OLED s'alimente en 3V3.

- Dans le fichier répertoire de votre projet STM32CudeIDE, ajouter la libraire *OLED* adaptée au microcontrôleur utilisé (NUCLEO-F303K8 ou STM32F072RBT6).
  - Copier les fichiers suivants dans le dossier Inc de votre projet STM32CubeIDE :
    - *fonts.h*
    - *ssd1306.h*
    - *test.h*
  - Copier les fichiers suivants dans le dossier Src de votre projet STM32CubeIDE :
    - *fonts.c*
    - *ssd1306.c*
    - *test.c*



3. Programme à faire pour utiliser la bibliothèque *OLED*,

- Modifier le fichier *main.c* afin d'inclure la bibliothèque *OLED* dans le programme principal après le commentaire `/* USER CODE BEGIN Includes */`

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
#include "fonts.h"
#include "test.h"
/* USER CODE END Includes */
```

- Ajouter les lignes de code suivantes dans le fichier *main.c* afin d'initialiser l'afficheur OLED:

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  SSD1306_Init();            // Initialisation de l'OLED
  while (1)
  {
      SSD1306_GotoXY(10, 0);                      // Go to 10,0
      SSD1306_Puts("HELLO", &Font_16x26, 1);      // Print "HELLO" with large font size
      SSD1306_GotoXY(10, 25);                     // Go to 10,25
      SSD1306_Puts("WORLD", &Font_11x18, 1);      // Print "WORD" with normal font size
      SSD1306_GotoXY(10, 50);                     // Go to 10,50
      SSD1306_Puts("SNEC", &Font_7x10, 0);        // Print "SNEC" with small font size
      SSD1306_UpdateScreen();                     // Update Screen

      HAL_Delay(2000);                            // Wait for 2 seconds

      SSD1306_Clear();                            // Clear the OLED Display

      SSD1306_GotoXY(10, 0);                      // Go to 10,0
      SSD1306_Putc(0x41, &Font_16x26, 1);         // Print character with the correspon-
ding hexadecimal code
      const uint8_t Donnee1='A';                  // Define a variable
      SSD1306_GotoXY(10, 25);                     // Go to 10,25
      SSD1306_Putc(Donnee1, &Font_16x26, 1);      // Print variable Donnee1
      SSD1306_UpdateScreen();                     // Update Screen

      HAL_Delay(2000);                            // Wait for 2 seconds

      SSD1306_Clear();                            // Clear the OLED Display

      /* USER CODE END WHILE */

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

Et voilà, le programme utilisant la bibliothèque OLED est opérationnel !

- Compiler le programme en mode Release.
- Téléverser le programme.
- Visualiser les caractères affichés sur l'écran OLED.

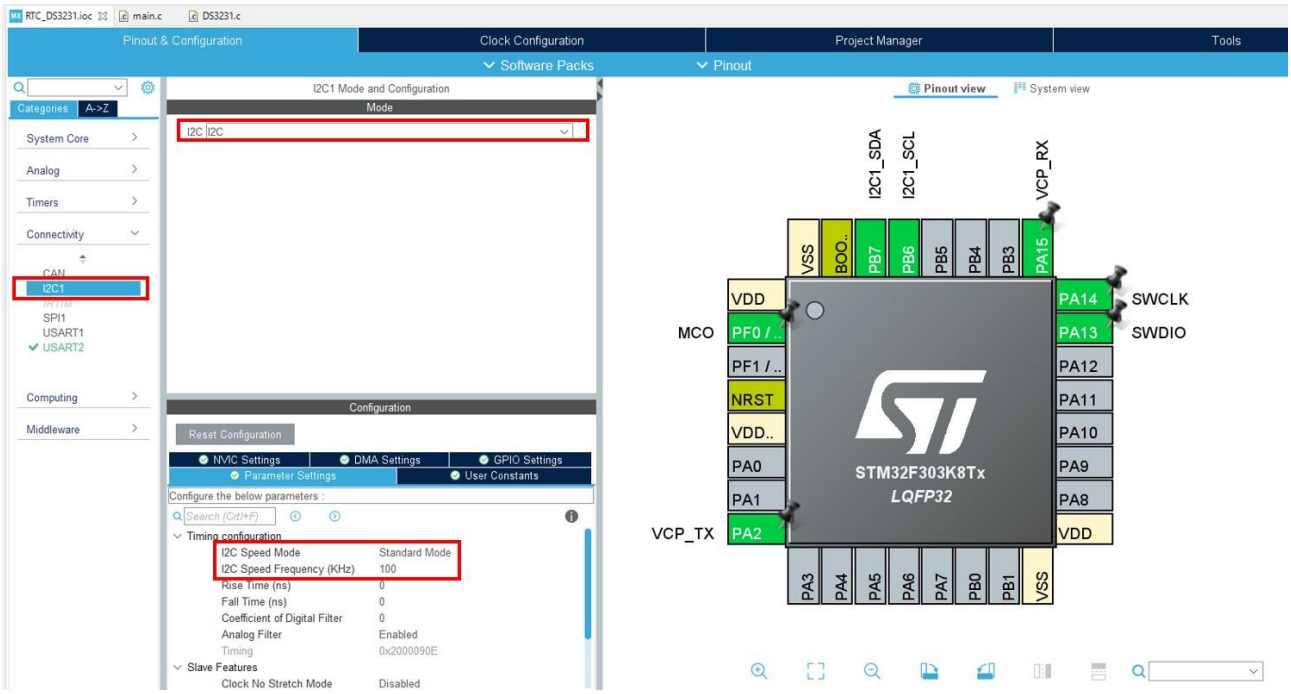
XIV. HORLOGE TEMPS RÉEL DS3231

L'objectif de cette partie est de présenter la programmation pour commander l'horloge temps réel de MAXIM INTEGRATED , DS3231.

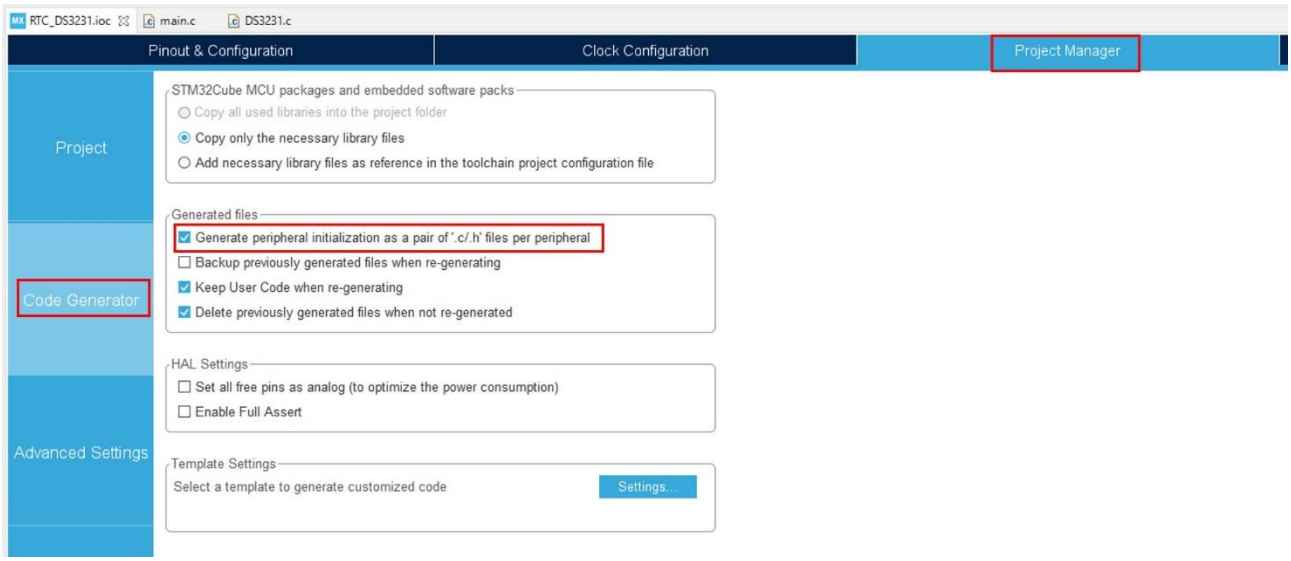
1. Installation de la bibliothèque *DS3231*

Pour pouvoir utiliser cette bibliothèque *DS3231*, la procédure à suivre est la suivante :

- *Sous le logiciel STM32CubeMX, création d'une liaison I2C en mode rapide (Fast Mode) à une fréquence de 400kHz.*



- Dans l'onglet Projet Manager => Code Generator, cocher la case « Generate peripheral initialization as a pair of '.h/.c' files per peripheral

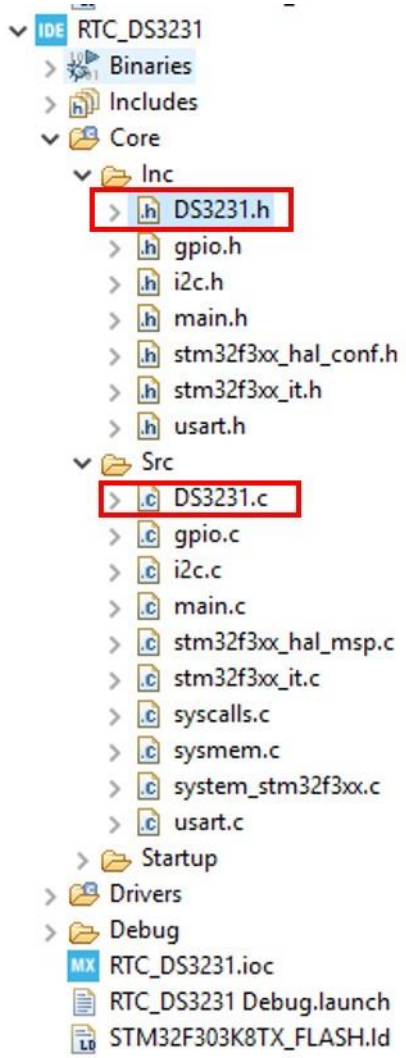


- Sauvegarder le projet afin de lancer la génération du code d'initialisation s'effectue.

- Câbler correctement l'horloge temps réel au microcontrôleur en fonction des broches identifiées.

- Dans le fichier répertoire de votre projet STM32CudeIDE, ajouter la libraire *DS3231* adaptée au microcontrôleur utilisé (NUCLEO-F303K8 ou STM32F072RBT6*)*.
    - Copier les fichiers suivants dans le dossier Inc de votre projet STM32CubeIDE :
        - *DS3231.h*

    - Copier les fichiers suivants dans le dossier Src de votre projet STM32CubeIDE :
        - *DS3231.c*



2. Programmation

- Modifier le fichier *main.c* afin d'inclure la bibliothèque *DS3231* dans le programme principal :

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "i2c.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "DS3231.h"
/* USER CODE END Includes */
```

L'instruction *Get_Time* permet de lire l'heure mesurée par l'horloge temps réel et de la stocker dans une variable appelée *time*.

- Ajouter les lignes de code suivantes afin de récupérer l'heure toutes les 2 secondes

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        Get_Time();            // Récupérer le temps réel et le stocker dans la va-
riable Time
        HAL_Delay(2000); // Attente de 2 secondes
    /* USER CODE END WHILE */

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

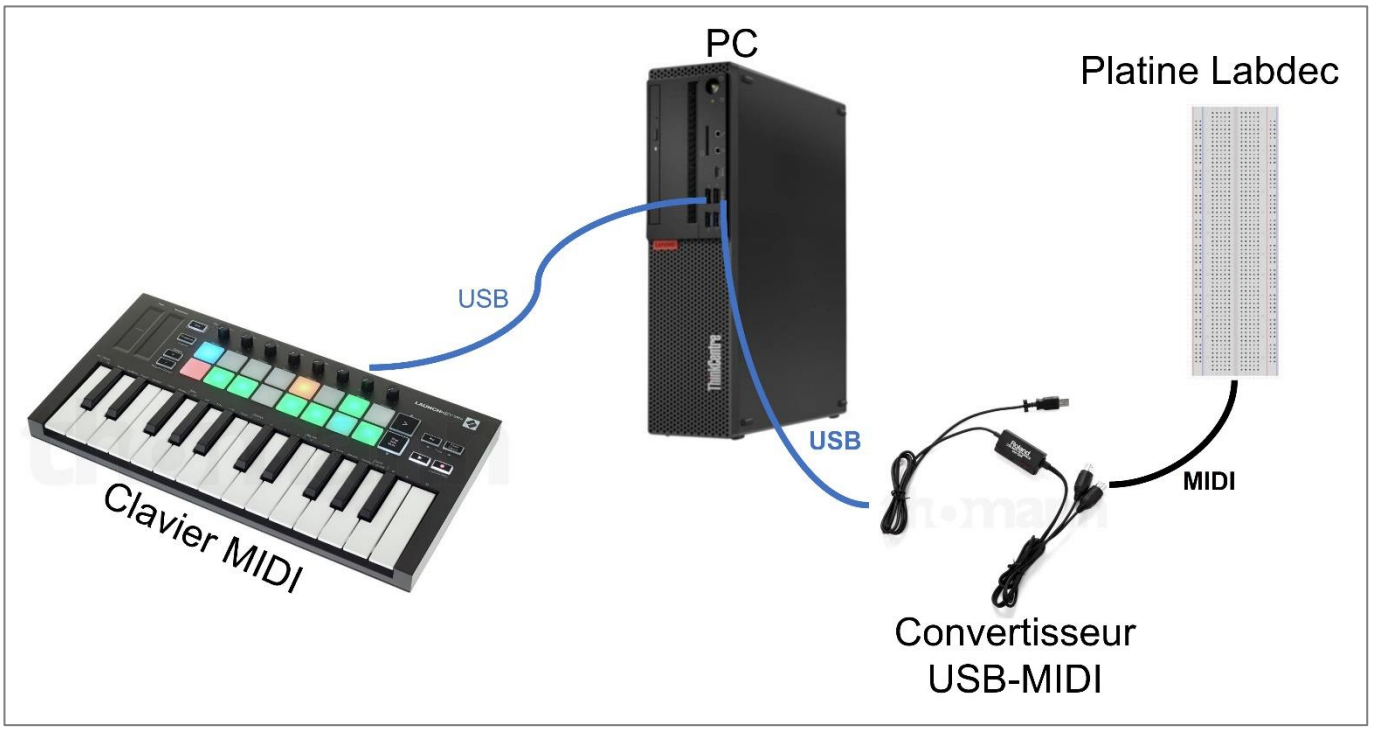
- Compiler puis débugger
- Visualiser la variable *time*

XV. PROTOCOLE MIDI

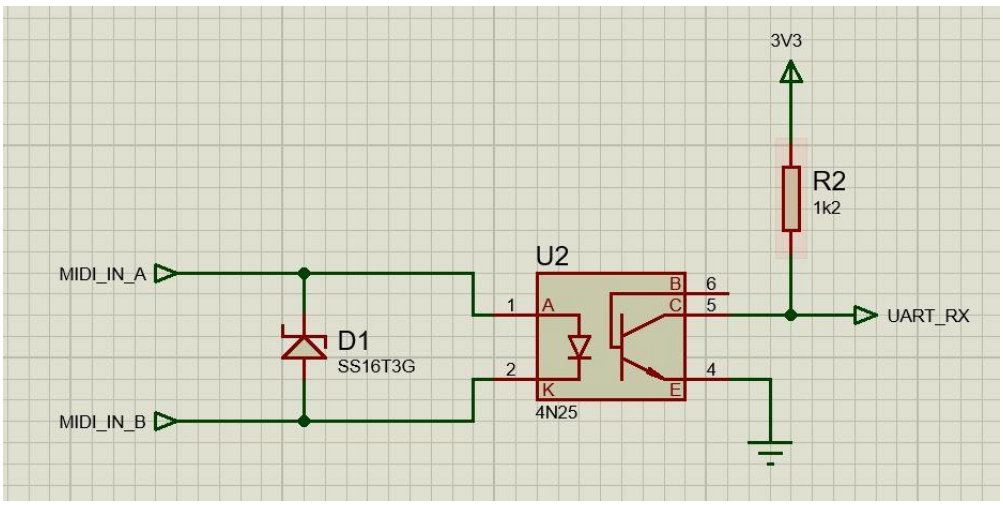
L'objectif de cette partie est de présenter la programmation pour réceptionner des trames provenant d'un clavier MIDI (exemple : Novation Launchkey MKII).

1. Câblage

Cette partie présente le câblage à effectuer pour relier le clavier MIDI à une carte NUCLEO-F303K8 connectée sur une platine Labdec.
Tout d'abord, il faut relier la clavier MIDI à la platine Labdec comme l'indique le schéma ci-dessous :



Sur la platine labdec, voici le schéma de câblage à effectuer pour accéder au signal UART_RX émis par le clavier MIDI.



Ce signal UART_RX est à relier la broche UART RX de la carte NUCLEO-F303K8.

2. Configuration de la liaison série

Une liaison série UART est à configurer sur la carte NUCLEO-F303K8. La liaison série est à configurer avec un débit de 31 250bps et en mode interruption.

3. Programmation

Une fois le code d'initialisation générée, voici les lignes de code à copier au début de votre programme vers la ligne 50 afin d'initialiser les variables nécessaires :

```
/* Private variables ---------------------------------------*/
UART_HandleTypeDef huart1 ;
UART_HandleTypeDef huart2 ;

/* USER CODE BEGIN PV */
uint8_t Data_RX[1] ;
uint8_t Data_RX_MIDI_NoteOFF[3] ;
uint8_t Data_RX_MIDI_NoteON[3] ;
int flag_NoteON ;
int flag_NoteOFF ;
uint8_t Inc;

/* USER CODE END PV */
```

Voici les lignes de code à copier dans la fonction *main* avant la boucle *while* afin d'initialiser les variables et d'activer la réception UART en interruption :

```
/* Initialize all configured peripherals */
MX_CPIO_Init() ;
MX_UART2_UART_Init() ;
MX_USART1_UART_Init() ;

/* USER CODE BEGIN 2 */

/* USER CODE END 2 */


/* Infinite loop */
/* USER CODE BEGIN WHILE */
Inc = 0 ;
flag_NoteON= 0 ;
flag_NoteOFF= 0 ;
HAL_UART_Receive_IT(&huart1, Data_Rx, 1) ;

While(1)
{

   /* USER CODE END WHILE */

   /* USER CODE BEGIN 3 */

   /* USER CODE END 3 */


}
```

Voici les lignes de code à copier dans votre programme vers la ligne 150 afin de programmer la réception UART en interruption des trames MIDI émises par le clavier MIDI.

```
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback (UART_HandleTypeDef *huart)
{
        if (Data_RX[0] == 0x90 || flag_NoteON == 1)
         {
             flag_NoteON = 1 ;
             Data_RX_MIDI_NoteON[Inc] = Data_RX[0] ;
             Inc ++ ;
             if (Inc == 3)
             {
                 Inc = 0 ;
                 flag_NoteON = 0 ;
              }
             else
             {
             }
         }
         else
         {
             Flag_NoteON = 0 ;
         }

        if (Data_RX[0] == 0x80 || flag_NoteFF == 1)
         {
             flag_NoteFF = 1 ;
             Data_RX_MIDI_NoteOFF[Inc] = Data_RX[0] ;
             Inc ++ ;
             if (Inc == 3)
             {
                 Inc = 0 ;
                 flag_NoteOFF = 0 ;
              }
             else
             {
             }
         }
         else
         {
             Flag_NoteOFF = 0 ;
         }

         HAL_UART_Receive_IT(&huart1, Data_RX, 1); // Activation de la réception UART par interruption
}
/* USER CODE END 4 */
```

- Compiler puis débugger
- Visualiser les variables *Data_RX_MIDI_NoteOFF et Data_RX_MIDI_NoteON*

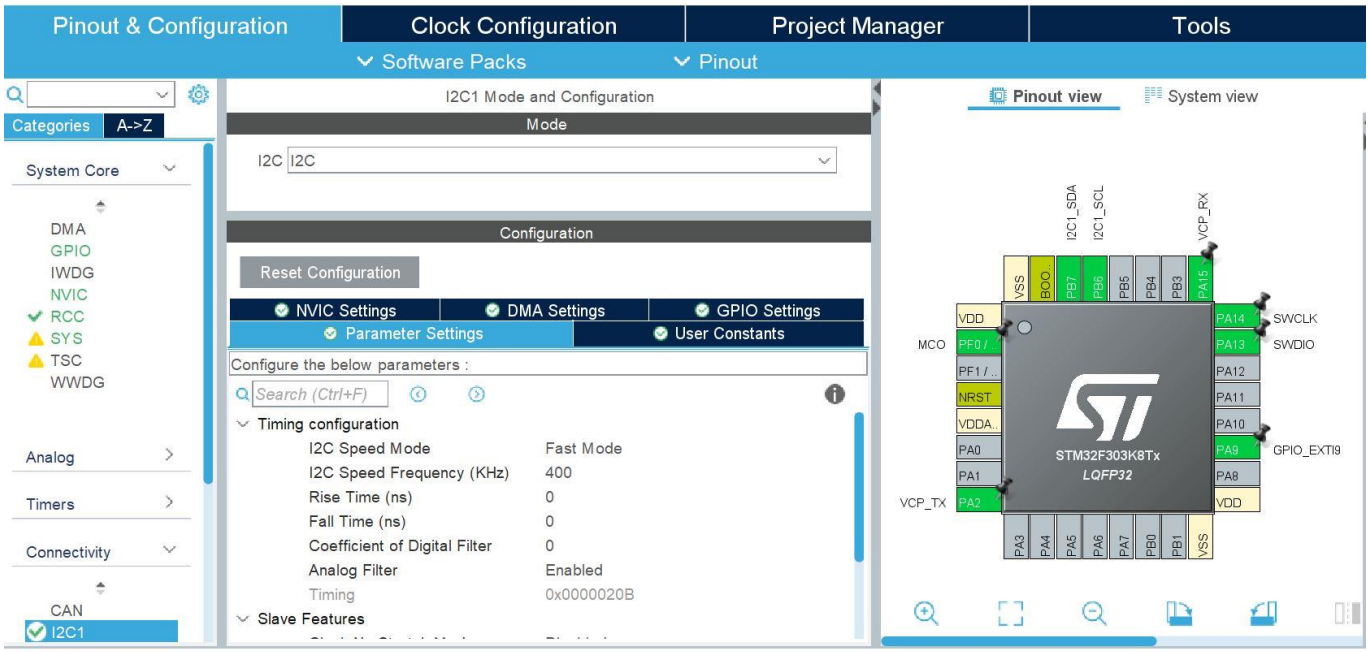
XVI. DÉTECTEUR DE MOUVEMENT APDS-9960

L'objectif de cette partie est de présenter la programmation pour commander le détecteur de mouvement de chez AVAGO, APDS-9960.

Source du programme : https://elektronikaembedded.wordpress.com/2018/02/16/interfacing-dfplayer-mini-dfr0299-mp3-player-module-with-stm32f103c8t6/

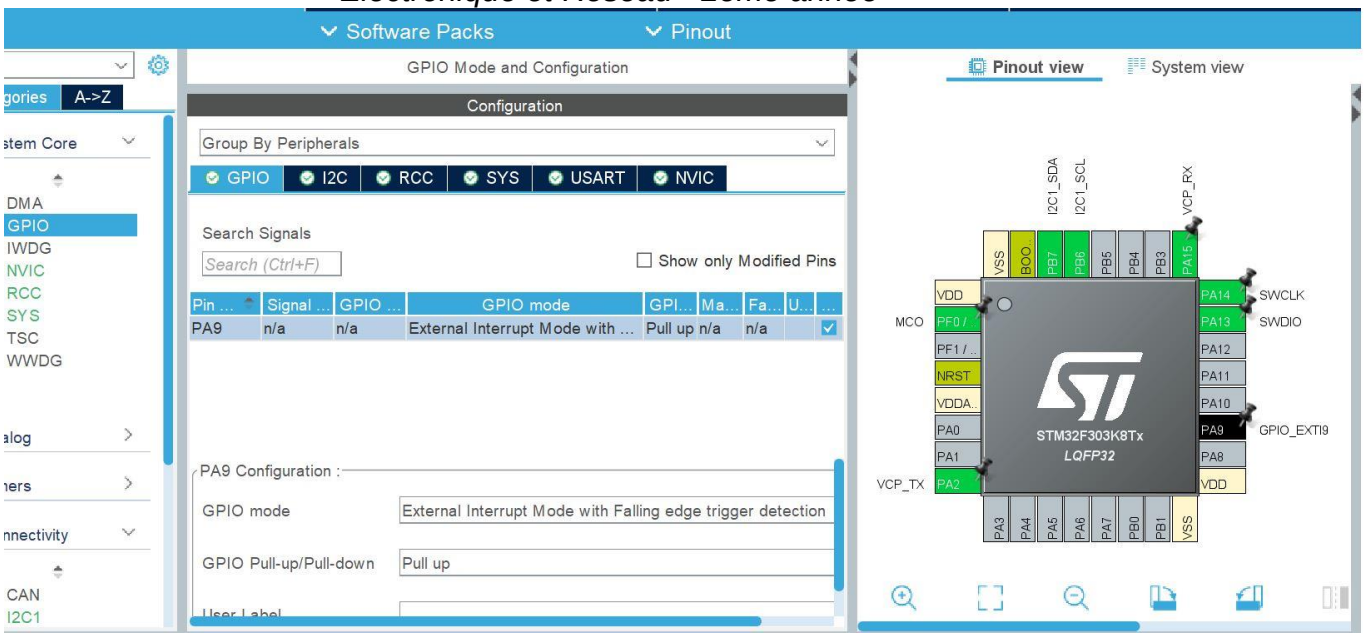
1. Installation de la bibliothèque *apd9960*

Pour pouvoir utiliser cette bibliothèque *apd9960*, la procédure à suivre est la suivante :



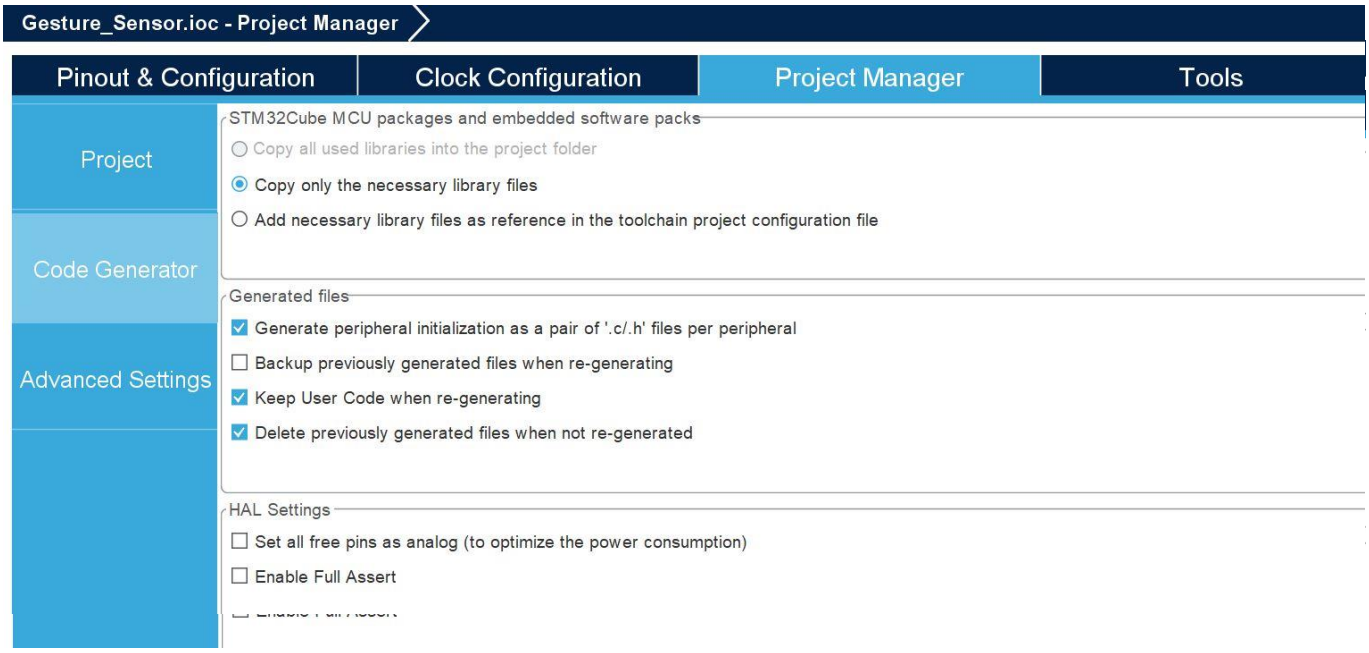
- *Sous le logiciel STM32CubeMX, création d'une liaison I2C en mode rapide (Fast Mode) à une fréquence de 400kHz.*

- *Sous le logiciel STM32CubeMDX, création d'une broche d'interruption (GPIO_EXTI) à configurer comme indiqué ci-dessous.*



- *Sous le logiciel STM32CubeMX, activer la prise en compte de l'interruption EXT dans l'onglet NVIC.*

- Dans l'onglet Projet Manager => Code Generator, cocher la case « Generate peripheral initiaization as a pair of '.h/.c' files per peripheral



- Sauvegarder le projet afin de lancer la génération du code d'initialisation s'effectue.

- Câbler correctement le détecteur de mouvement au microcontrôleur en fonction des broches identifiées. Bien penser à relier la broche d'interruption

- Dans le fichier répertoire de votre projet STM32CudeIDE, ajouter la libraire *apd9960* adaptée au microcontrôleur utilisé (NUCLEO-F303K8 ou STM32F072RBT6*)*.
    - Copier les fichiers suivants dans le dossier Inc de votre projet STM32CubeIDE :
        - *apd9960.h*
        - *typedef.h*

    - Copier les fichiers suivants dans le dossier Src de votre projet STM32CubeIDE :
        - *apd9960.c*
        - *typedef.c*

4. Programme à faire pour utiliser la bibliothèque *apd9960*

- Modifier le fichier *main.c* afin d'inclure la bibliothèque *apd9960* dans le programme principal :

```
16   ******************************************************************************
17   */
18 /* USER CODE END Header */
19 /* Includes ------------------------------------------------------------------*/
20 #include "main.h"
21 #include "i2c.h"
22 #include "usart.h"
23 #include "gpio.h"
24
25 /* Private includes ----------------------------------------------------------*/
26 /* USER CODE BEGIN Includes */
27 #include "apd9960.h"
28 #include "typedef.h"
29 /* USER CODE END Includes */
30
31 /* Private typedef -----------------------------------------------------------*/
32 /* USER CODE BEGIN PTD */
33
34 /* USER CODE END PTD */
```

```
44 /* USER CODE END PM */
45
46 /* Private variables ---------------------------------------------------------*/
47
48 /* USER CODE BEGIN PV */
49 volatile int Gesture_Flag;
50 int gesture;
51 uint8_t id;
52
53 /* USER CODE END PV */
54
```

```
  /* USER CODE BEGIN 2 */

  gesture = 0;
  //uint8_t CONFIG_REG=0x92;
  //HAL_I2C_Master_Transmit(&hi2c1, 0x39<<1, &CONFIG_REG, 1, 500);  // Read ID register 0x92
  //HAL_I2C_Master_Receive(&hi2c1, 0x39<<1, &id, 1, 500);               // Store in id variable


  apds9960init();
  HAL_Delay(500);

  enableGestureSensor(true);
  HAL_Delay(500);

  ////////////////////////////////
  // TO BE CONTINUED //////////////
  ////////////////////////////////
```